VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky

Bakalářská práce

2012 Lukáš Kubíček

VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Testování systémů v praxi
System Testing in Practice

2012 Lukáš Kubíček

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání bakalářské práce

Student: **Lukáš Kubíček**

Studijní program: B2647 Informační a komunikační technologie

Studijní obor: 2612R025 Informatika a výpočetní technika

Téma: Testování systému v praxi
System Testing in Practice

Zásady pro vypracování:

Cílem práce je vytvořit popis a netriviální ukázky využití metodik a technik testování softwarových systémů v praxi se zaměřením na fázi testování systému a podpůrné nástroje pro vytváření testů, provádění testů, zprávu incidentů a konfigurační management.
Práce se zaměří na oblasti:
1. Tvorbu testovacích dat.
2. Systémové testování.
3. Automatizaci systémového testování.
4. Výkonnostní testování.

Práce bude rovněž obsahovat netriviální ukázky využití testovacích nástrojů pro výše zmíněné oblasti testování, především ukázky skriptů pro jednotlivé nástroje. Práce by měla obsahovat popis způsobu použití nástrojů jako jsou HP Quality Center, Jira, SOAPui, Selenium IDE, Selenium Webdriver, QTP a Jmeter.

Seznam doporučené odborné literatury:

Steven R.Rakitin: Software Verification and Validation for Practitioners and Managers, Second Edition, ISBN: 1-58053-296-9
Dále dle pokynu vedoucího bakalářské práce.

Formální náležitosti a rozsah bakalářské práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí bakalářské práce: **Ing. David Ježek, Ph.D.**

Datum zadání: 18.11.2011
Datum odevzdání: 04.05.2012

doc. Dr. Ing. Eduard Sojka
vedoucí katedry

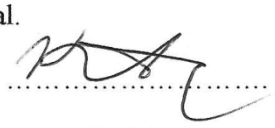
prof. RNDr. Václav Snášel, CSc.
děkan fakulty

Prohlášení studenta

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně.

Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

Dne: 9.4.2012



Podpis

Abstrakt

Tato bakalářská práce se snaží popsat vybrané nástroje z oblasti testování počítačových nebo webových aplikací. V práci popisuji několik vybraných nástrojů z několika oblastí testování. Práce se nezaměřuje na podrobný popis jednotlivých nástrojů, ale snaží se o vyzdvihnutí silných stránek, nebo poukazuje na slabší vlastnosti. Nastiňuji zde také důležitost a některé metody pro tvorbu testovacích dat. Cílem práce je nastínit možnosti, které máme dnes k dispozici, a které se zabývají testováním.

Klíčová slova

Testování, automatické testy, regresní testy, testovací data, selenium, webdriver, výkonnostní testování.

Abstract

This bachelor thesis attempts to describe selected tools from area of software applications. The work descibes several tools from several areas of software testing. The work does not focus on a detailed description of each tool but tries to highlight the strenghts and points the weak features. Also outline the importance and some methods for creating the test data. The goal of this work is to outline the options which we have avaiable for software testing.

Key words

Tetsing, automatisation tests, reggression tests, testing data, selenium, webdriver, performance testing.

Obsah

1 Úvod

Cílem bakalářské práce Testování systémů v praxi je popsat jednotlivé nástroje, které se v dnešní době využívají v různých oblastech testování. Pro jednotlivé oblasti jako je systémové, automatické nebo výkonnostní testování existuje celá řada podpůrných nástrojů. Tyto nástroje se více či méně snaží pokrýt co nejširší okruh využití, který ne vždy je efektivní v celém rozsahu. Z tohoto důvodu se v této práci se zaměříme na nejsilnější stránky jednotlivých nástrojů, jejich použití v praxi, a pokud to bude možné tak i ukázat na praktických ukázkách. Nesmíme ani zapomenout na podpůrné nástroje používané během celého životního cyklu vyvíjeného systémů. Jsou to nástroje, které spravují vstupní požadavky systému a přehledně monitorují aktuální stav vývoje. V této práci si popíšeme „Quality center“ od společnosti Hewlett Packard nebo „Jira“ od Atlassian. Tyto nástroje jsou velice komplexní a vzhledem k tématu této bakalářské práce si je nastíníme jen zběžně.

V první části se letmo zmíníme o testování obecně a nastíníme něco málo o Configuration managementu.

V druhé části se zaměříme na jeden z důležitých úkolů při jakémkoliv testování. A tím není nic jiného než získávání nebo generování testovacích dat. Bez korektních dat by nebylo možné s jistotou říct, že daný test proběhl v pořádku či nikoliv.

Ve třetí části se zaměříme na nástroje používané při automatickém testování. Těchto nástrojů je celá řada a tak vyberu ty nejvíce využívané. Budou to, Selenium IDE, Selenium WebDriver a z placených nástrojů to bude HP QTP (Quick Test Profesional). Stranou ponechám nástroje jako například Test Complete nebo IMacro a další.

V další části se budeme věnovat nástroji „Jmeter“, který se využívá pro výkonnostní testování. A také nezapomeneme ani na SOAPui, který je nenahraditelný pomocník při testování komponent s využitím SOAP nebo REST rozhraní.

V poslední části budeme srovnávat jednotlivé nástroje a zpracování přehledu, který by měl pomoci při rozhodování kdy jednotlivé nástroje použít. A jak s nimi začít pracovat.

Obsahem této bakalářské práce není popisovat do podrobností jak s jednotlivými nástroji pracovat. Tato problematika je již podrobně popsána a zdokumentována na stránkách výrobců. Cílem této práce je vyzdvihnout jednotlivé nástroje v určitých oblastech testování systémů a dopomoct tak snazšímu rozhodnutí, zdali daný nástroj je vhodný pro vybrané testování.

2 Testování

2.1 Co je testování

Položme si otázku co to vlastně testování je, k čemu slouží a je opravdu tak důležité, když aplikaci vytvářejí vývojáři? Ano, testování je opravdu důležité a patří k nedílné součásti při vývoji softwaru. Tým lidí, kteří provádějí testování je zodpovědný za konečnou výstupní kvalitu dané aplikace a je zárukou, že vyvíjený software splňuje minimálně požadovanou kvalitu určenou zákazníkem. Uvědomme si, že ani ten nejlepší vývojář není schopen naprogramovat bezchybnou aplikaci. Ne ovšem z důvodu, že by nebyl dost dobrý, ale prostě jen proto, že není nezaujatý vůči svému kódu a často ani neví jak má celá aplikace fungovat jako celek. K tomu zde slouží testovací tým, který se snaží nezávisle otestovat aplikaci, tak jak ji bude používat zákazník a jeho uživatelé. Na druhou stranu je třeba si uvědomit, že software jsme schopni otestovat jen do určité úrovně, která by měla být specifikována akceptačními kritérii. To znamená, že v praxi neexistuje bezchybný software. Například při testování vstupního formulář, nějaké aplikace, která má několik rozbalovacích seznamů, textových polí atd.… V případě, že bychom chtěli otestovat všechny kombinace, dostáváme se tak k obrovským číslům a takové testování se pak stává mnohem dražším než samotný přínos. Z tohoto důvodu je důležité mít kvalitně specifikovaná akceptační kritéria, která určují minimální kvalitu funkčnosti, ať už pro celou aplikaci tak pak jako dílčí kritéria pro jednotlivé iterace vývoje. A nezapomeňme na jedno základní pravidlo, které říká, že čím později je chyba nalezena tím, dražší je její odstranění.

2.2 Rozdělení testování

Testování se dá rozdělit na dvě základní kategorie.

- **Funkční testování** (Functional testing) se zabývá, jak už je v názvu zmíněno funkcionalitou systému. A mělo by odpovědět na otázku „může uživatel udělat to či ono a funguje daná funkcionalita?“

- **Nefunkční testování** (Non-functional testing) se zabývá vlastnostmi systémů, které nesouvisí s funkcionalitami, ale spíše s chováním, které práci se systémem ovlivňuje. Například, stabilita nebo zatížitelnost

Testování můžeme dále rozdělit do několika úrovní

- **Unit testing** je nejčastěji psáno přímo vývojáři, nejčastěji k otestování nějaké konkrétní funkce v kódu. Tento typ testů nemá ověřit funkčnost systému nebo komponenty, ale pouze elementární funkcionality.

- **Integration testing** slouží k otestování rozhraní jednotlivých komponent a komunikaci mezi nimi.

- **Regression testing,** kterým ověřujeme, zda systém funguje stejně i po úpravě kódu a to buď opravou některé chyby, nebo implementací nové funkcionality. Regresní testy by měly být součástí každého releasu a potvrzuji, že daná změna nevytvořila jinou na stávající funkčnosti.

- **System testing** testuje software jako celek a zaručuje se za jeho výstupní kvalitu. Samotné systémové testování se dá dále dělit na mnoho dalších podskupin jako například:

    - **Perfomance testing,** kdy ověřujeme stabilitu a chybovost systému při určité zátěži.

    - **Security testing** ověřuje, zda systém dostatečně chrání data a zároveň je zachována požadovaná funkčnost.

    - **Error guessing** dává člověku, který systém testuje jistou volnost a spoléhá na jeho úsudek a zkušenosti. Tento typ testování se snaží odhalit chyby, které se nemusí najít při regresním testování.

    - **Destructive testing** je zaměřen na pokus o vyvolání chyby zadáváním úmyslně nekorektních vstupních dat. Při tomto testování se snažíme ověřit chování systému při vyvolané chybě. Ošetřování výjimek, mapování chybových zpráv atd…

Více informací najdete v publikacích [1, 2, 3].

3 Software configuration management

SCM se zabývá nástroji a technikami, které nám umožní spravovat vyvíjený software a následné změny. SCM se nezabývá ovšem pouze nástroji a procesy, ale také řeší otázky hardwarového zajištění vývojového prostředí nebo jednotlivých prostředků pro členy týmu. SCM dále řeší verzování systému a jednotlivých komponent, správu změn a udržování záznamů o tom jaké změny byly provedeny. V kostce se tedy jedná o kompletní správu vývoje a údržbu vývoje, díky níž jsou následné změny rychle a efektivně proveditelné a tím i mnohem levnější. Několik následujících úloh SCM vycházejí z Best practices.

Identifikace částí systému pro verzování a uložení – jde o správné určení toho jaké části systému, budou verzovány a kde budou verze uloženy. Úložiště by mělo být nejlépe jedno centrální, které shromažďuje verze jednotlivých komponent a za použití některého nástroje pro sestavení, například maven, můžeme jednoduše nakonfigurovat tu, kterou verzi komponenty budeme využívat. S tím souvisí i kontrola a revize verzí a systému. Systém se ve většině případů rozrůstá a je třeba například nové komponenty zahrnout do systému verzování. Jednotlivé verze systémů by měly být funkční a každé sestavení pro produkci by mělo mít svou verzi a hlavně by mělo být funkční a otestované. Poté není problém se kdykoliv vrátit k předchozí funkční verzi. Například implementujeme novou funkcionalitu a po tom co ji zákazník začne používat, se rozhodne, že ji nechce. V případě, že máme dobře vedený SCM je taková změna časově náročná jen, tak jak dlouho trvá udělat nové sestavení. Není třeba nic měnit v kódu aplikace. Další důležitou součástí je využívání některého z verzovacích systémů zdrojových kódů, jako například distribuovaný systém GIT. Tento systém má jedno centrální úložiště a každý z vývojářů pracuje na svém vlastním klonu. Slévání s centrálním klonem má následně na starost správce centrálního úložiště. Díky tomu může více vývojářů dělat na jedné části aplikace a zároveň máme zajištěnou bezpečnost dat proti poškození. I v případě, že dojde ke ztrátě všech dat z centrálního úložiště tak se nic neděje, protože každý kdo se systémem pracuje, má na svém lokálním počítači svůj klon, který je kopií centrálního klonu. Samozřejmě, že GIT není jediný nástroj, je to jen jedna možnost z několika.

Další věc, kterou configuration management řeší je dokumentace procesů a jednotlivých součástí aplikace. Týká se to jak dokumentace vývojářské, tak dokumentace pro testování nebo pro jednotlivá prostředí. Dokumenty by měly být aktuální vzhledem ke změnám, které ve vývoji dané aplikace nastaly.

Z toho všeho vyplývá, že správně vedený SCM nám zaručuje flexibilitu při vývoji softwaru, rychlou implementaci změn, rychlou opravu chyb, přehledy o provedené práci a práci, kterou je třeba ještě udělat. To vše má za následek zvýšení efektivity a snížení nákladů. A minimálně toto je důvod se nad otázkami SCM zamyslet před tím, než se vrhneme bezhlavě do vývoje nového systému. Bez SCM bychom se dříve či později dostali do stavu, kdy již nebudeme schopni efektivně plnit přání zákazníka.

Podrobnější informace ohledně SCM jsou k dispozici v publikacích [4, 5].

4 Testovací data

4.1 Motivace

Vstupní data pro testování jsou základním kamenem každého testu, protože bez nich nejsme schopni daný test provést a už vůbec ne vyhodnotit. Na kvalitě testovacích dat závisí ne jen schopnost test vůbec provést, ale hlavně test správně vyhodnotit. Vstupní data pro testy obecně by měly, v co možná nejvyšší míře odrážet reálná data se kterými budou pracovat koncoví uživatelé. Je také vhodné se zamyslet nad tím, pro jaký typ testů data potřebuji. Například pro systémové testování bude vhodné mít údaje někde, kde je mám okamžitě k dispozici, nebo odkud si je můžu jednoduše kopírovat a vkládat do příslušných polí v testované aplikaci. V tomto případě nám může skript vypsat údaje třeba na konzoli. Úplně jiný případ jsou automatické nebo výkonnostní testy. U těch nepotřebujeme data nikde kopírovat, ale chceme je načítat ze souboru. V takovém případě je třeba mít skript, který vytvoří požadovaný datový soubor. Vytvářením skriptů pro vytváření testovacích dat sice zabereme nějaký čas, ale v konečném součtu nám může vhodně napsaný skript ušetřit hodiny a hodiny práce.

4.2 Obecné informace

Před tím než začneme psát skript, který by nám generoval testovací data, je třeba se zamyslet nad několika věcmi**. Jak často a kolikrát budeme data potřebovat?** V případě, že data použijeme jen jednou, na specifický test bude asi zbytečné si psát program na vytvoření těchto dat. To ovšem neplatí v případě, že pro jeden test potřebuji větší kvantum specifických dat, které bychom manuálně vytvářeli příliš dlouho. **Jak chci výsledné data prezentovat?** Jestli je potřebuji na manuální testování a stačí je mít vypsané na monitoru, nebo je potřebuji mít uložené někde v souboru. **Jak vyřeším konfiguraci?** Toto téma není tak triviální jak se může na první pohled zdát. Testovací prostředí bývá často velice flexibilní, kdy na serverech běží několik verzí ať už webových služeb nebo databází. A často dochází pouze k rekonfigurování jednotlivých komponent. Z tohoto důvodu je třeba mít i naši konfiguraci skriptu vhodně řešenou, kdy například změna databáze v testovacím prostředí pro nás znamená změnu jednoho řádku v konfiguračním souboru nebo třídě. Tím zajistíme snadnou použitelnost skriptu a značnou úsporu času. **Jak udělat dokumentaci?** Jelikož týmy se neustále mění, jedni pracovníci přicházejí a jiní odcházejí tak i těchto podpůrných skriptů je vhodné vyřešit zdokumentování, vhodně okomentovat kód atd.… A hlavně potřebuji vůbec vytvářet skripty pro výrobu testovacích dat nebo si postačím s dostupnými data generátory, o kterých se zmíním jen letmo. Při generování dat ovšem nesmíme zapomínat na metodiky testování, protože nejde o to vytvořit hromadu dat, ale mít data, která v co největším záběru otestují danou aplikaci. To znamená, že stejně tak jako validní data potřebujeme i data, u kterých očekáváme zápornou reakci systému. Takže při vytváření systému je třeba vzít v úvahu metodiky testování a dle nich si i data připravovat. Testovací nástroje samy o sobě tyto

metodiky neimplementují, ale pouze zpracovávají naše vstupní data. Proto, a teď se budu opakovat, je velice důležité před samotným vytvořením dat si rozmyslet jaké a kolik dat vlastně potřebujeme.

4.3 Generování náhodných dat

V této kapitole se zaměřím na generování náhodných dat. Takto vytvořená data potřebujeme v případě, že testovaný systém vyžaduje jako vstupní údaje data, která musí odpovídat různým vnitřním předpisům. Jinými slovy námi testovaný systém očekává specifický formát vstupních údajů a ve většině případů má i nastaven různé validace vstupu. Takovými daty můžou být například obecné údaje specifické pro určitou zemi nebo region. Mezi takové určitě patří třeba rodné číslo, které musí být v určitém formátu a ještě navíc obsahuje kontrolní mezisoučet. Z více specifičtějších to je například IMEI mobilního telefonu, které stejně jako rodné číslo má pevně daný kontrolní součet. Dále jsou to data specifická pro systém, který testujeme. Jestli mám uvést příklad, tak to může být například login studenta, který bude obsahovat tři písmena z příjmení a určitý počet číslic. Takových pravidel může mít systém několik. I u těchto údajů bude vyžadována specifická validace vstupu. Z tohoto důvodu je zapotřebí, aby i naše testovací data splňovala tato pravidla. Je naprosto jasné, že pro drtivou většinu těchto hodnot musíme mít napsaný skript, nebo mít k dispozici aplikaci, která nám tato data vygeneruje. Jakým způsobem budeme data reprezentovat, záleží na požadavcích testů.

4.4 Získání existujících dat

Získáním existujících dat mám na mysli použití údajů, které jsou již v systému uloženy. Ve většině případů se jedná o databázi. Jestliže pracujeme na systému, který již funguje v produkci, potom je ideální, zkopírovat produkční data do testovacího prostředí. Tím zajistíme, že můžeme pro testování používat reálná data, která budou odrážet reálné požadavky koncových uživatelů. Nesmíme ovšem zapomenout na právní legislativu dané země ve vztahu vůči ochraně osobních údajů a jejím nakládáním s nimi. K tomu nám často pomůže degradace dat, při jejich kopírování z produkční databáze. Při této změně dat je nutné dát pozor na to, aby výsledná pozměněná data byla ještě použitelná pro další testování. Čím déle je systém nasazen v produkci a čím více uživatelů s ním pracuje, tím větší je naše šance najít zde pro náš test hodnotu, která přesně odpovídá požadavkům testu. Musíme si ovšem uvědomit, že ani tato data nemusí obsahovat potřebné informace, které potřebujeme k našemu testu. Vezměme v potaz, že testujeme novou funkcionalitu nebo konfiguraci nové služby. V takovémto případě se námi potřebná data v produkci ještě vyskytovat nemohou. Je tady ještě jedna věc, nad kterou je třeba se zamyslet a tím je konzistentnost dat v testovacím a produkčním prostředí. V případě, že se daný systém neustále vyvíjí, může brzy nastat situace, kdy kombinace údajů v našem prostředí neodpovídá reálným hodnotám, se kterými pracují uživatelé. Například nabízený produkt již neexistuje a nemůže se stát, že by s ním zákazník pracoval. V našem testovacím prostředí se ale nachází, a při jeho použití v testu nám může nepříjemně pozměnit

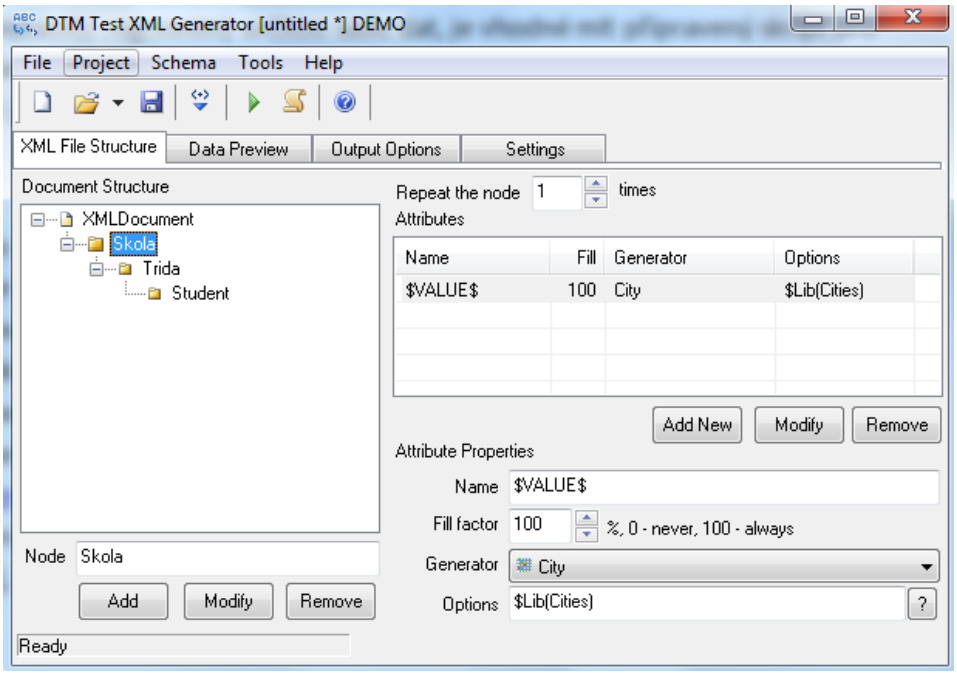
výsledky. Z tohoto důvodu je potřeba a nazvěme to Best Practice, v určitých časových intervalech odstranit z testovacího prostředí všechna data a nahradit je aktuálními daty z produkce. Interval těchto aktualizací závisí na mnoha faktorech, které jsou specifické pro daný projekt. Hodně záleží na četnosti změn a závažnosti změn, které uvolňujeme do produkce. Pro získávání dat z databází nám mohou posloužit SQL dotazy spouštěné přímo v databázovém klientovi nebo pomocný skript, který udělá tuto práci za nás.

4.5 Vkládání nových dat

V mnoha případech potřebujeme pracovat s daty, která v systému jsou, ale vzhledem k testování nového systému nebo nové funkcionality nemůžeme tato data nahrát z produkce. Nezbývá nic jiného, než si je vytvořit a do databáze či jiného úložiště je nahrát. V případě, že se jedná o databázi, můžeme data vkládat přímo přes databázového klienta nebo pomocí námi napsaného skriptu. Musíme si však dát veliký pozor na dodržení konzistentnosti dat v tabulkách. Například vkládáme-li nového studenta do tabulky student, musíme si být jistí, že neexistuje jiná tabulka, která by měla informace o studentovi obsahovat. Ať už z důvodů závislostí nebo redundantních dat. V takovém případě by náš test nemusel proběhnout správně, nebo by byl výsledek zkreslený. Je třeba si uvědomit, že jeden formulář v uživatelském rozhraní může být rozložen do několika databázových tabulek. V jiném případě může systém data před uložením pozměnit, přepočítat apod. Než začneme vkládat nové záznamy do systémové databáze, je třeba provést podrobnou analýzu toho, jaká data v databázi potřebujeme mít a kde všude se tato data v databázi nacházejí. V případě, že budeme vkládat do databáze opakovaně záznam pro testování, je vhodné si pro tento případ vytvořit pomůcku, která nám vkládání záznamů do databáze usnadní. Může se jednat o předpřipravený SQL dotaz, anonymní proceduru nebo napsaný skript. Pro případ, že potřebujeme k pravidelným testům (nejčastěji regresním) určitou sadu dat, je vhodné mít připravený skript pro vytvoření všech těchto dat. Ušetříme tím mnoho práce v případě aktualizací databáze na nová produkční data, nebo při jiném znehodnocení testovacích dat.

4.6 Data generátory

V dnešní době již existuje celá řada nástrojů pro generování náhodných testovacích dat. A může se jednat o placené nástroje nebo volně dostupné. Tyto nástroje dnes již pokrývají oblasti jak vytvoření data a jeho uložení do souboru libovolného formátu nebo mohou uložit data přímo do databáze. Jeden z nástrojů je například portál generatedata [6], u kterého je možné generovat data z webového prohlížeče a není nutná žádná instalace. Například samotné vývojové prostředí Visual studio nabízí test data generátor, jehož popis naleznete na [7]. Nebo placené nástroje sqledit [8], kde máme k dispozici jak nástroje pro práci s databázemi tak například nástroj DTM XML Generator, dostupný na stránkách sqledit [8], který vygeneruje náhodné xml, dle vašeho návrhu nebo dle nahraného schématu. V práci je využitá volně dostupná demoverze, v případě potřeby plné funkcionality je třeba software zakoupit.



Obrázek 1 - ukázka softwaru DTM XML Generátor

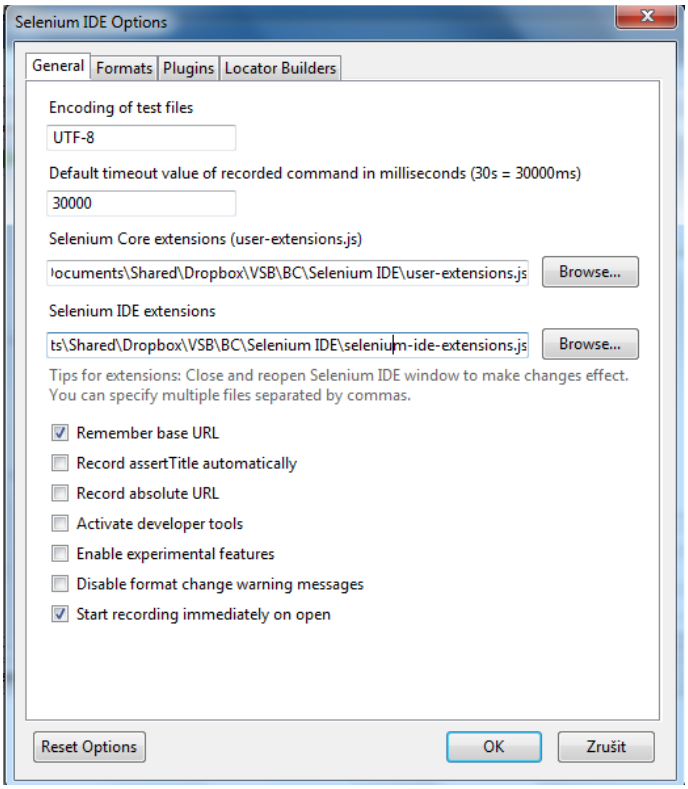
5 Nástroje pro automatické testování

5.1 Motivace

Kdy je vhodné přemýšlet o automatických testech? Už ze samotného názvu je zřejmé, že síla automatických testů je tam, kde opakujeme stejný test pořád dokola. Ve většině případů to jsou právě regresní testy, kdy potřebujeme ověřit, zda nová funkcionalita neovlivnila tu stávající. Cílem automatizovaného testu není zkoumání, zda momentální stav odpovídá momentálnímu stavu specifikace, ale zjištění, zda se chování od předešlého testu změnilo či nezměnilo. Než se rozhodneme pro automatické testy, je třeba si promyslet, jestli čas strávený psaním a údržbou testů nebude mnohem větší než opakované manuální testování. Protože si musíme uvědomit, že při využití automatického testu nejde pouze o jeho napsání, ale také údržbu, která je v průměru 30% času práce u plně alokovaného automatického testera.

5.2 Selenium IDE

Selenium IDE (Integrated Developmen Enviroment) slouží k záznamu a vytváření Selenium skriptů pro automatizované testování. Tento nástroj je integrovaný jako doplněk do prohlížeče FireFox a slouží pro testování webových aplikací. Mohlo by se tedy zdát, že Selenium IDE je pro nás naprosto nepoužitelný, v případě, že potřebujeme testovat aplikaci v Internet Exploreru, Chrome či jiném prohlížeči. Je sice pravda, že jej přímo nemůžeme použít v těchto prohlížečích, ale vzhledem k jeho funkcím lze prokázat, že tomu tak není. Selenium IDE, používá místo programovacího jazyka pro vytváření skriptů svůj pseudo kód a tím pádem se právě proto stává vhodnou volbou testovacího nástroje. Nevyžaduje totiž po testerech znalost programovacích jazyků jako jiné nástroje například Visual Bacis Skript u HP QTP, nebo Java, Ruby, C# či Python u WebDriveru. Na první pohled po nainstalování by se mohlo zdát, že postrádá několik základních funkcí jako je například nahrávání testovacích dat z externích souborů, vytváření smyček nebo podmínek. Selenium IDE ovšem umožňuje vložení externích knihoven, které musí být napsány v Java Skriptu. Vložením několika těchto knihoven, viz obrázek 2, získáme možnost nahrávat data z externích souborů, například csv a procházet je v cyklu po řádcích. Další vymoženosti jsou právě zmíněné smyčky, podmínky a mnohé jiné. Vhodné knihovny, které nám zajistí všechny funkce, které jenom můžeme potřebovat pro automatické testování jsou : datadriven.js, flowcontrol.js, user-extensions.js jako Selenium Core Extension. A Selenium IDE extension : selenium-ide-extensions.js. Tyto knihovny se přidávají v nabídce „Options“ nástroje, jak je ukázáno na obrázku 2. Nelze přidávat více knihoven najednou, ale opětovným přidáváním je musíme přidat postupně.



Obrázek 2 – vložení knihoven do Selenium IDE

Tvar pseudo kódu je složen ze tří následujících hodnot. *Command* je příkaz, který říká, co budeme provádět jako například *Click*, *Pause*, *Select* apod. Druhý příkaz *Target* udává lokalizaci elementu na webové stránce. A poslední *Value* není povinný a slouží například jako hodnota vkládána do textového pole nebo rozbalovacího seznamu. Další věc, kterou je nutné znát před samotným začátkem psaní skriptů je práce s proměnnýma. Můžeme si vytvořit vlastní proměnné a to příkazem *store* jako je na obrázku níže, kdy do proměnné *myVar* ukládám hodnotu 234. Viz obrázek 3.

Command	Target	Value
store	234	myVar

Obrázek 3 – vytvoření proměnné

Následné zavolání proměnné se provede takto : *${myVar}*. Takže například pro vložení mé hodnoty do textového pole se provede následující příkaz. Viz obrázek 4.

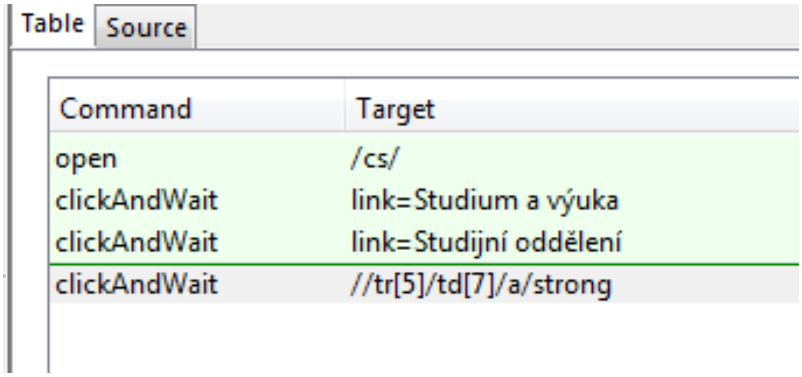
Command	Target	Value
store	234	myVar
type	textBox ID	${myVar}

Obrázek 4 – použití proměnné

5.2.1 Záznam

Selenium IDE, nám dává dvě možnosti vytvoření testovacího skriptu. Na nás je rozhodnutí, která z následujících možností je pro nás ta nejvhodnější.

První možnost je záznam, kdy pouze procházíme testovanou aplikaci tak, jak chceme, aby se skript později sám prováděl, a nástroj nám sám zaznamenává jednotlivé kroky. Tento způsob vypadá na první pohled velice jednoduše, ale vyžaduje po nahrání manuální průchod jednotlivými kroky a jistou korekci. Samotný skript je po nahrání bohužel nepoužitelný. Je potřeba upravit lokalizace elementů na stránkách, protože nástroj se sám rozhoduje, který je pro něj nejvhodnější a často se stává, že nevybere ten správný způsob. Zkusme si tedy nahrát malý skript, který na stránkách VŠB klikne na Studium a výuka, Studijní oddělení a poté na kontakty FEI. Výsledný záznam, by mohl vypadat následně viz obrázek číslo 5:



Obrázek 5 – ukázka nahraného testu v Selenium IDE

Již druhý příkaz při následném spuštění neprovede to, co očekáváme. Příkaz *clickAndWait* klikne na daný link, ale očekává otevření nové stránky. To se však nestane, jelikož po kliknutí na tento link se pouze změní obsah stávající stránky. Musíme tedy změnit tento a stejně tak následující příkazy z *clickAndWait* na *click*. Další věc se týká již zmíněných lokalizací. Když použijeme přesnou lokalizaci elementu podle Xpath *//tr[5]/td[7]/a/strong*, tak bude tento způsob fungovat vždy jen za předpokladu, že daný objekt bude stále na stejném místě. Často ovšem obsah webových aplikací bývá dynamický a proto tento způsob lokalizace může při spuštění testu selhat. V tomto případě by naše oprava mohla vypadat například takto: *//a[contains(@href, `kontakty-fei`)]*. Touto změnou zajistíme, že se klikne na správný link, ať už bude kdekoliv. Stačí jen aby url obsahovala „kontakty-fei“. Poslední změnou je přidání krátkých čekacích příkazů, protože chceme, aby na kliknutí došlo až po jeho zobrazení. Takže výsledný upravený skript by mohl vypadat asi takto, jak je ukázáno na obrázku číslo 6.

Command	Target
open	/cs/
pause	1000
click	link=Studium a výuka
pause	1000
click	link=Studijní oddělení
pause	2000
click	//a[contains(@href, 'kontakty-fei')]

Obrázek 6 – ukázka pseudokódu po úpravě

Z toho vyplývá, že samotný záznam je sice jednoduchý a rychlý, ale vzhledem k následné úpravě nemusí být vždy nejvhodnější. A to jsme se ani nepokoušeli o práci s cykly a proměnnýma. V tomto případě je náš záznam jen málo užitečný a budeme se zaměřovat spíše na manuální psaní skriptu.

Druhá možnost je manuální zápis. Tento způsob vytváření automatických skriptů nám umožňuje značnou variabilitu při tvorbě. Mohou to být již zmíněné funkce jako nahrávání testovacích dat z externího souboru, používání smyček, podmínek a práce s proměnnýma. Důležitou součásti je lokalizace elementů na stránce a ve většině případů může za následný pád testů právě nevhodná lokalizace. Nejčastěji používáme pro nalezení prvku na stránce nějakého jednoznačného identifikátoru. V mnoha případech postačí atribut *Id* nebo *name* a v případě linků to bývá link. Ne vždy tyto atributy stačí, protože často jsou webové aplikace komplexnější a bývají poskládány z mnoha menších částí. Často je tak Id objektu generováno náhodně použitým frameworkem a tudíž lokalizace právě podle toho parametru vede k pozdějším chybám. Nebo například v případě, že chceme najít v tabulce výsledků určitý záznam. V tomto případě je vhodnější použít Xpath, který umožňuje najít prvek na stránce mnoha způsoby. Začněme ale od začátku.

Na následujícím obrázku číslo 8, je malá ukázka skriptu, který nahraje data z csv souboru a prochází je ve smyčce tolikrát, kolik má řádků bez hlavičky. Hlavička v souboru definuje názvy proměnných. Takže například test, který bude načítat data ze souboru, viz obrázek 7, provede test dvakrát a v textu mám k dispozici dvě proměnné *${username}* a *${password}*, aniž bychom je museli před tím jakkoliv přiřazovat příkazem *store*. Navíc nám příkaz *nextTestData* načte do proměnných aktuální hodnotu z řádku, na kterém test zrovna je.

username	password
kub0086	test
kub0087	test

Obrázek 7 – výpis z csv souboru

Takže test pro tyto data by mohl vypadat asi takto, jak je ukázáno na obrázku číslo 8.

Command	Target	Value
loadTestData	file://C:/Temp/01_seleniumIDEdata.csv	
while	!testdata.EOF()	
nextTestData		
type	//input[contains(@id, 'userName')]	${username}
type	//input[contains(@id, 'password')]	${password}
click	//input[contains(@id, 'logIn')]	
waitForPageToLoad		5000
pause	3000	
verifyTextPresent	You has been successfully logged in. Please select y...	
goBack		
type	//input[contains(@id, 'userName')]	
type	//input[contains(@id, 'password')]	
endWhile		

Obrázek 8 – ukázka průchodu dat z scv souboru ve smyčce

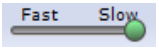
Další užitečnou věcí jsou zmíněné podmínky. Na následující ukázce si ukážeme jak například procházet tabulkou, která zobrazuje jen určitý počet záznamů tak dlouho než najdeme námi hledaný záznam nebo nebudeme na konci tabulky. Viz obrázek číslo 9.

Command	Target	Value
click	//a[contains(@href, 'propertyList.xhtml')]	
pause	10000	
loadTestData	file://C:/Temp/04_seleniumIDEdata.csv	
nextTestData		
loop table and searchour first prop id		
delete property for first added person		
while	true	
storeElementPresent	//td[contains(., '${propertyId}')]	isPresent
gotoIf	${isPresent} ==true	DELETE
storeElementPresent	//input[contains(@id, 'next_true')]	isEnabled
gotoIf	${isEnabled} ==true	END
click	//input[contains(@value, 'next')]	
pause	20000	
endWhile		
label	DELETE	
click	//td[contains(., '${propertyId}')]/../td[7]/a	
pause	5000	
label	END	
verifyTextPresent	Record has been deleted	

Obrázek 9 – ukázka cyklů s podmíněným ukončením v Selenium IDE

V tomto příkladě hledáme v tabulce náš záznam s hodnotou, která je uložena v proměnné *propertyId*. Ovšem nevíme, kolikrát musíme listovat v tabulce, která zobrazuje jen určitý počet záznamů. Je zde vidět načtení hledané proměnné ze souboru a následně vytvoření nekonečné smyčky za pomocí *while true*. Poté zjišťujeme, jestli hledaný element na stránce je zobrazený a výsledek příkazu *storeElementPresent*, který je true nebo false, uložíme do proměnné *isPresent*. Následně vytváříme podmíněný odskok, který je reprezentovaný příkazem *gotoIf*. Místo odskoku je definované názvem nějakého labelu. Takže v případě, že hledaný element se na stránce nachází, vyskočí test ven ze smyčky a bude pokračovat na labelu s názvem *DELETE*. Poté se pokračuje ve všech příkazech, které tento label následují. V případě, že k nalezení nedojde, jsme již na konci seznamu a to je v našem příkladě reprezentováno hodnotou *next_true* v Id objektu, přeskočí se mazání a vzhledem k tomu, že systém nic nesmazal tak i poslední řádek vyhodí chybu testu. Kromě příkazu *storeElementPresent* můžeme použít i *storeAttribute* a mnoho jiných. Cílem této práce není popsat jednotlivé příkazy a jejich použití, ale spíše nastínit syntaxi tohoto nástroje. Zmíním se zde ještě o jednom problému a tím jsou rozbalovací seznamy. Pro klasický rozbalovací seznam je možné použít pouze příkaz *select* a jako hodnotu mu předat *value*, které chceme vybrat. V dnešní době je ale mnoho klasických prvků překryto grafickým objektem, na který již příkaz *select* fungovat nebude. V tomto případě nám zbývá pouze na tento objekt kliknout a poté znova kliknout na prvek v rozbaleném seznamu.

5.2.2 Spouštění testů

V selenium IDE, můžeme samozřejmě vytvářet jak testovací případy tak celé testovací sady, který v sobě obsahuje libovolný počet testovacích případů. Před samotným spuštěním je doporučováno snížit rychlost testů, čímž omezíme případné chyby, kdy je test o mnoho rychlejší než interakce testované aplikace. Snížení rychlosti se provede posuvníkem 

. Poté můžeme spustit aktuální testovací případ  nebo celou testovací sadu  . V případě že jeden test zhavaruje, začne se provádět následující. V případě verifikací výsledků dojde pouze k zčervenání verifikace, ale test pokračuje dále.

5.2.3 Záznam kódu a export

Selenium IDE, zaznamenává svůj pseudokód jako HTML a uložený test si můžeme otevřít a prohlédnout i ve webovém prohlížeči. V případě, že by někomu nevyhovovalo grafické rozhraní nástroje, tak má možnost psát testy přímo v HTML. Selenium IDE navíc poskytuje možnost převést váš test do unit testu a to hned do několika jazyků. Máme tedy možnost si v tomto nástroji test vytvořit, převést do unit testu a po malé úpravě použít pro libovolný prohlížeč. Díky této možnosti se nám může tento nástroj hodit i v případě, že budeme testovat aplikaci například v Chrome. Samozřejmě to nelze bez základní znalosti programovacích jazyků a následné úpravy testu. Protože překlad, nám přeloží základní operace obsažené v Selenium IDE, ale již není schopen přeložit operace v námi importovaných knihovnách. V takovémto případě vypíše do těla testu chybovou zprávu, tudíž

po exportu testu s logováním bude výsledný unit test v Javě vypadat asi takto, jak je zobrazeno na obrázku číslo 10.

```
@Test
public void test01Login() throws Exception {
        // ERROR: Caught exception [unknown command [loadTestData]]
        // ERROR: Caught exception [unknown command [while]]
        // ERROR: Caught exception [unknown command [nextTestData]]
        driver.findElement(By.xpath("//input[contains(@id, 'userName')]")).clear();
        driver.findElement(By.xpath("//input[contains(@id, 'userName')]")).sendKeys("${username}");
        driver.findElement(By.xpath("//input[contains(@id, 'password')]")).clear();
        driver.findElement(By.xpath("//input[contains(@id, 'password')]")).sendKeys("${password}");
        driver.findElement(By.xpath("//input[contains(@id, 'logIn')]")).click();
        // ERROR: Caught exception [ERROR: Unsupported command [isTextPresent]]
        driver.navigate().back();
        driver.findElement(By.xpath("//input[contains(@id, 'userName')]")).clear();
        driver.findElement(By.xpath("//input[contains(@id, 'userName')]")).sendKeys("");
        driver.findElement(By.xpath("//input[contains(@id, 'password')]")).clear();
        driver.findElement(By.xpath("//input[contains(@id, 'password')]")).sendKeys("");
        // ERROR: Caught exception [unknown command [endWhile]]
        // stay logged as last user
        driver.findElement(By.xpath("//input[contains(@id, 'userName')]")).clear();
        driver.findElement(By.xpath("//input[contains(@id, 'userName')]")).sendKeys("${username}");
        driver.findElement(By.xpath("//input[contains(@id, 'password')]")).clear();
        driver.findElement(By.xpath("//input[contains(@id, 'password')]")).sendKeys("${password}");
        driver.findElement(By.xpath("//input[contains(@id, 'logIn')]")).click();
}
```

Obrázek 10 – ukázka exportu kódu z nástroje Selenium IDE

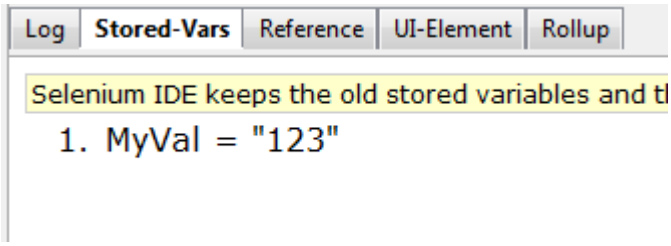
5.2.4 Vhodné doplňky

Pro Selenium IDE, existuje celá řada doplňků (plug-in), které si můžeme do tohoto nástroje integrovat. Vyzdvihneme zde jen pár nejužitečnějších.

HighLight Elements: Jedná se o doplněk, který během průchodu testu žlutě vyznačuje na webových stránkách, s kterým elementem právě pracuje. Tím vám pomůže odhalit, že jste někde například špatně lokalizovali objekt na stránce. 

Power Debugger: Doplněk po zapnutí zastaví procházený test v případě chyby a my tak máme možnost se podívat, co způsobilo daný problém v aktuálním kroku testu. 

Stored Variables Viewer: Jeden z velkých pomocníků, který aktuálně zobrazuje hodnoty všech použitých proměnných. Viz obrázek číslo 11.



Obrázek 11 – doplněk zobrazující hodnoty proměnných

5.2.5 Závěr

Selenium IDE, patří mezi často používané nástroje pro automatizované testování a to právě díky tomu, že znalost programovacího jazyka, zde není podmínkou, ale výhodou. Navíc je tento nástroj volně ke stažení a není třeba platit za drahé licence.

5.3 Selenium WebDriver

Selenium WebDriver je open source nástroj používaný na automatické testování webových aplikací. Na rozdíl od Selenium RC, již nemusíme myslet na spouštění RC serveru před každým testem. Pro psaní automatického testu v tomto nástroji využíváme WebDriver API. Webdriver následně volá přímo specifické webové prohlížeče a využívá jejich integrovaných nástrojů pro automatické testy. Pro psaní našich testů můžeme využít několik programovacích jazyků Java, C#, Ruby, Python, PHP nebo Perl. Nejvíce využívaný s největší podporou je Java a proto jsme ukázku testu udělali právě v něm. Největším přínosem takto psaných automatických testů je možná integrace s testovacím prostředím. Na rozdíl, od Selenium IDE není nutností mít předpřipravený zdroj dat. Vzhledem k tomu, že test píšeme přímo v programovacím jazyce, můžeme mít v tomto testu použité třídy a metody, které si samy v případě potřeby dané testovací data obstarávají. Pro psaní takového testu využíváme některého z vývojářských nástrojů, jako jsou Eclipse, NetBeans apod. Následně máme několik přístupů k testu. Můžeme jej napsat jako unit test a poté jej i jako unit test spouštět a využitím assert metod. Nebo jej napíšeme jako samostatnou aplikaci, ve které si sami specifikujeme ověřování správnosti provedení testu. Při psaní je třeba dodržovat několik zásad, které zajistí lepší stabilitu daných testů. Dejme tomu, že testujeme určitou část webové aplikace, která obsahuje větší množství operací. Tento celý test se nazývá testovací sada a je to vlastně sada jednotlivých testovacích případů. Dohromady tvoří jeden celek, ale provádějí se postupně. Je třeba dodržovat co největší izolovanost jednotlivých testovacích případů v rámci jedné testovací sady. Následný testovací případ by neměl být závislý na výsledku předchozího. V případě, že použijeme závislost tak riskujeme pád celého testu, i když chyba byla třeba pouze v prvním testovacím případě. Dále je nutné promyslet a vytvořit vhodné ošetřování výjimek, případné logování chyb do souboru nebo výpisem na monitor. Při logování chyb je vhodné oddělit havárii testu od nesplněné podmínky správného výsledku. Je třeba rozlišit a řádně zapsat do výsledků jestli test neprošel z důvodu například výpadku sítě, nebo pouze výsledek se lišil od očekávaného.

V našem testu testujeme webovou aplikaci PropertyRecord a celý test se skládá z pěti jednotlivých testovacích případů. Zaměříme se nyní na jednotlivé části.

Konfigurace: V běžné praxi se nepoužívá pouze jedno testovací prostředí, ale několik. V našem případě jsme použili tři standardně využívané. ST – systémové testování, AT – akceptační testování a PT – produkční testování. Každé prostředí mívá jinou konfiguraci, využívá jinou databázi apod. Z tohoto důvodu je vhodné mít pro každé prostředí určitý set nastavení a před spuštěním testů pouze specifikujeme, ve kterém prostředí test spouštíme. Ukázka možného způsobu konfigurace je vidět na obrázku 12 a 13. Je to jen jeden možný způsob z mnoha, protože můžeme použít property soubor, xml konfiguraci apod.

```
public class Configuration {
    public static final String BROWSER = "firefox"; // firefox, ie, chrome
    public static final String CSV_SEPARATOR = ";";
    public static final String ENV = "st";

    public static final HashMap st = new HashMap();
    static {
        st.put("CONNECTION_URL", "jdbc:derby://localhost:1527/PropertyRecord;user=property;password=property");
        st.put("errorLog", "C:\\Temp\\erorLog_ST_");
        st.put("assertionLog", "C:\\Temp\\assertionLog_ST_");
        st.put("testData", "C:\\Temp\\data.csv");
        st.put("app_address", "http://localhost:22809/PropertyRecordVis/");
    }
```

Obrázek 12 – ukázka možné konfigurace testovacích prostředí

```
public static final HashMap enviroment = new HashMap();
static {
    enviroment.put("st", st);
    enviroment.put("at", at);
    enviroment.put("pt", pt);
}
```

Obrázek 13 – ukázka možné konfigurace testovacích prostředí

Object Repository: Je to vlastně knihovna prvků, které budeme v naší aplikaci používat. Výhoda použití této knihovny je v tom, že máme všechny objekty na jednom místě a v případě změny některého prvku stačí provést opravu jen na jednom místě, jak ukazuje obrázek 14.

```
//menu
public static final String MENU_NEW_LOCATION = "//a[contains(., 'New comp. location')]";
public static final String MENU_NEW_PERSON = "//a[contains(., 'New Person')]";
public static final String MENU_NEW_PROPERTY = "//a[contains(., 'New Property')]";
public static final String MENU_PROPERTY_LIST = "//a[contains(., 'Property list')]";
public static final String MENU_SEARCH = "//a[contains(., 'Search property')]";
```

Obrázek 14 – příklad možné knihovny objektů

Pro lokalizaci objektu ve WebDriver máme několik možností. Můžeme použít lokalizaci pomocí ID, name, XPATH nebo obsaženého linku. V mém případě jsme použili XPATH.

Zdroj dat: Jako zdroj dat jsme použili csv soubor, který obsahuje potřebné testovací data. Jednoduše bez nutnosti konfigurace tak určíme, kolikrát chceme test spustit. Je totiž pravděpodobné, že budeme potřebovat test spustit vícekrát s různými vstupními daty. V csv souboru potom každý řádek představuje jedno spuštění.

Externí knihovna: Pro opakované operace ve více testech je dobré si vytvořit knihovnu s těmito operacemi a do testu ji pouze importovat. V našem příkladě je knihovna použita pro načtení dat z csv souboru a pro logování chyb do souboru.

Main: spouštěcí metoda obsahuje instanci driveru pro specifický prohlížeč jak je vidět na obrázku číslo 15.

```
if (Configuration.BROWSER.equals("firefox")) {
    driver = new FirefoxDriver();
} else if (Configuration.BROWSER.equals("ie")) {
    driver = new InternetExplorerDriver();
} else if (Configuration.BROWSER.equals("chrome")) {
    driver = new ChromeDriver();
}
```

Obrázek 15 – ukázka specifikace prohlížeče ve WebDriver

Následně metoda pouze volá jednotlivé testovací případy, kterým předá tento driver jako parametr. V našem případě jsme záměrně nedodrželi úplnou izolovanost jednotlivých kroků, protože provádím všechny kroky nad jednou instancí driveru. Důvod této implementace je prostý, za cenu úplné izolovanosti jsme znatelně zvýšili rychlost testů. Vytvoření nové instance je časově náročná operace a tak při větším počtu kroků lze tímto způsobem zkrátit dobu, po kterou test běží. Testovali jsme rychlost vytváření instancí tak, že jsme desetkrát po sobě vytvořili novou instanci, načetli domovskou stránku a instanci zavřeli. Průměr na jednu instanci vyšel 10,3 sekundy. Tento test ovšem nezapočítává opakované přihlašování, které bychom museli před každým novým krokem provést. V mém případě by výsledné číslo bylo 15 sekund násobeno počtem testovacích případů. Vždy, je třeba volit mezi perfektně napsaným testem a reálnými možnostmi.

Jednotlivé kroky: uvnitř každého kroku již voláme funkce driveru a provádíme dané operace na testované aplikaci viz obrázek číslo 16.

```
public class Login implements TestCase{

    public void run(WebDriver driver, HashMap testData, ErrorLogger toError, int cycle) {
        try {
            WebElement query = driver.findElement(By.xpath(ObjectRepository.LOGIN_LOGIN));
            query.sendKeys(((ArrayList<String>)testData.get("login")).get(cycle));
            query = driver.findElement(By.xpath(ObjectRepository.LOGIN_PASSWORD));
            query.sendKeys(((ArrayList<String>)testData.get("password")).get(cycle));
            query = driver.findElement(By.xpath(ObjectRepository.LOGIN_SUBMIT));
            query.click();
            driver.manage().timeouts().implicitlyWait(5, TimeUnit.SECONDS);

            // assertion login
            try {
                query = driver.findElement(By.xpath(ObjectRepository.LOGIN_ASSERT));
            } catch(Exception ass) {
                toError.assertionFailed("Login failed", ass);
            }
        } catch (Exception e) {
            toError.toErrorLog(null, e);
        }

    }
}
```

Obrázek 16 – ukázka kroků ve Webdriver

WebDriver je dnes nejvyužívanější open source nástroj pro automatické testování. Jeho použití je vhodné zejména, kvůli velké základně uživatelů a tím i značnému množství diskusních fór, na kterých většinou najdeme odpověď na vzniklý problém.

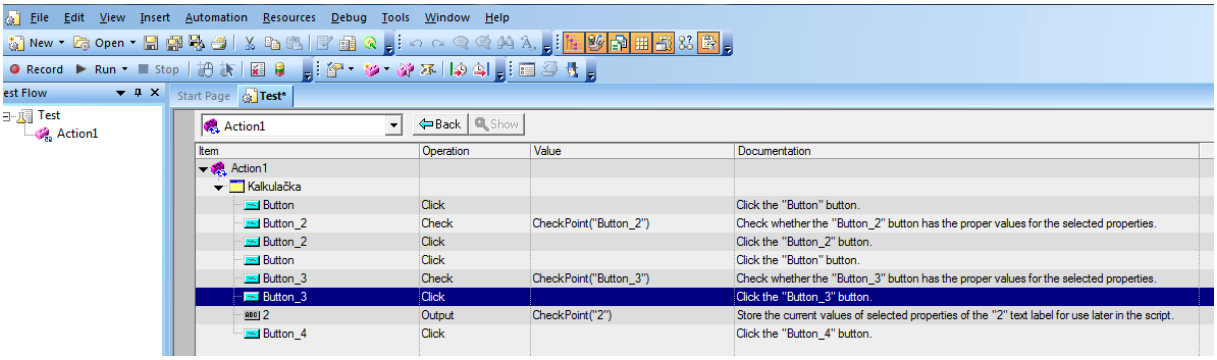
Kompletní informace k nástroji Selenium IDE a WebDriver jsou k dispozici, na stránkách [9]

5.4 HP QTP

Na tento nástroj se zaměříme o něco méně než na předchozí Selenium. QTP je licencovaný nástroj a již při prvním použití jde poznat, že vývoji se věnovali opravdu pečlivě. Jediná věc, která tento nástroj znatelně znevýhodňuje, je jeho cena. Jedná se o velice rozsáhlý nástroj, který by sám svým obsahem pokryl téma jedné bakalářské práce. QTP umí stejně dobře vytvářet automatické testy pro webové aplikace jako pro desktopové aplikace. To je oblast testování, kterou za použití Selenium nejsme schopni nahradit. Jako skriptovací jazyk se používá Visual Basic Skript, ale znalost tohoto jazyka není pro vytváření testů podmínkou. QTP umí nahrávat testy a zde je třeba podotknout, že o něco lépe než Selenium, ale i tak je třeba test projít a upravit podle našich požadavků. V případě, že budeme potřebovat vytvořit složitější testy s cykly a podmínkami, tak se bez základní znalosti programovacího jazyka neobejdeme. U tohoto nástroje bych vyzdvihl funkci Object repository. Jedná se o knihovnu elementů používaných při testech. Jak už jsem se o tom zmínil v kapitole WebDriver, tak je vhodné mít objekty, které voláme nejlépe na jednom místě. QTP k tomu nabízí možnost takovouto knihovnu sdílet a tím usnadňuje týmovou práci. Při následném vytváření testů již nemusíme znova a znova

lokalizovat elementy na stránce nebo objekty v aplikaci. Jednoduše použijeme knihovnu objektů, která je již vytvořena v rámci týmu.

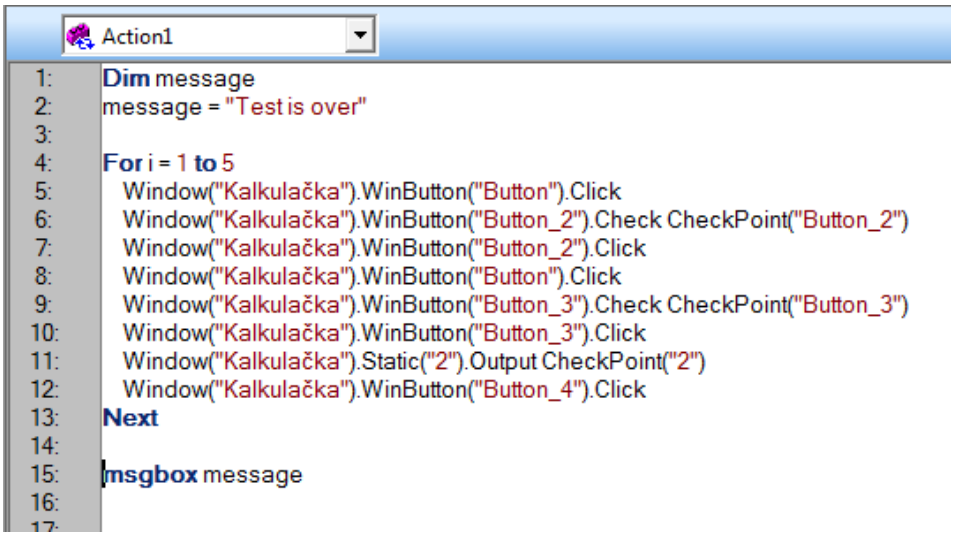
Vytvoření testu – po nahrání testu například windows kalkulačky by mohl test, po doplnění verifikací, vypadat asi takto viz obrázek číslo 17.



Obrázek 17 – ukázka testu v nástroji HP QTP

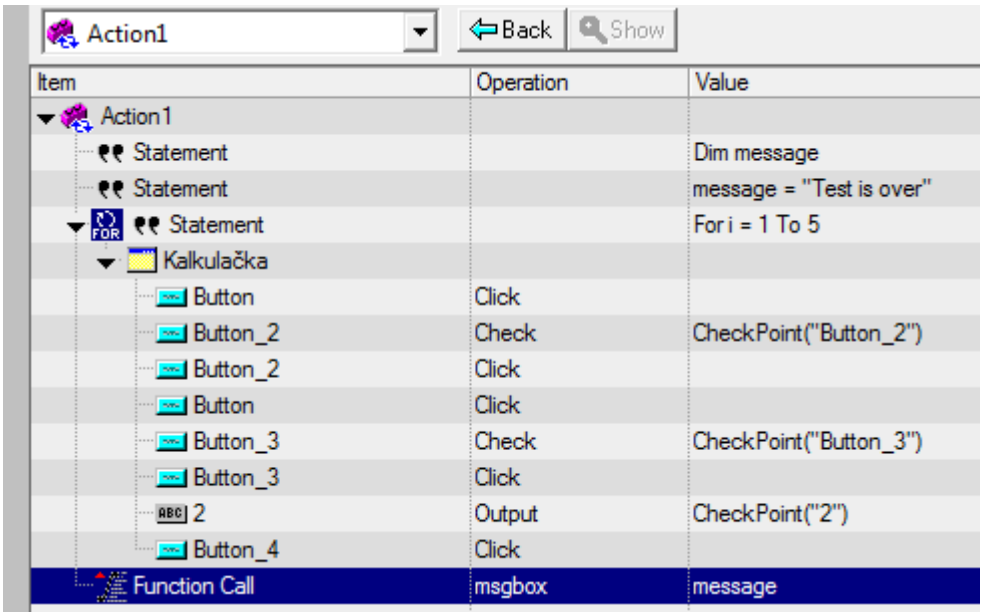
Na obrázku jde vidět grafický záznam testu, kdy kliknutím na jakoukoliv část můžeme modifikovat parametry, akci nebo validační kritéria. Ty jsou v QTP označeny jako *Checkpoint*, jednotlivé kontrolní body mají posléze vliv na výsledný výstup, ale ten si ukážeme později. Nyní si můžeme na liště dole přepnout z grafického zobrazení testu na expertní. Jedná se o zápis ve VBS. Tam můžeme provádět pokročilé modifikace testů, vytvářet iterace, podmínky, validace apod.

Po přidání iterací ve VBS a doplnění testu o konečnou zprávu o tom, že test doběhl do konce, bude kód vypadat takto, jak je ukázáno na obrázku číslo 18.



Obrázek 18 – ukázka testu v VBS editoru

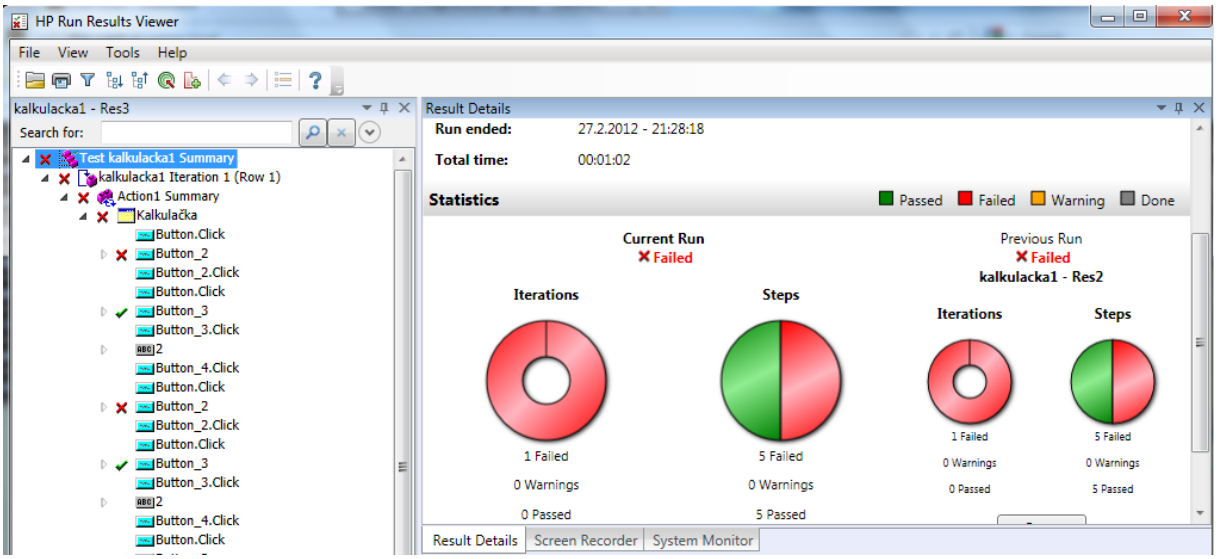
Při zpětném přepnutí do grafického zobrazení uvidíme, že změny se okamžitě promítly i zde. A z toho lze vyvodit, že úpravy v kódu se daly udělat i v grafickém zobrazení viz obrázek 19. Je už otázkou uživatele, co je pro něj výhodnější a rychlejší.



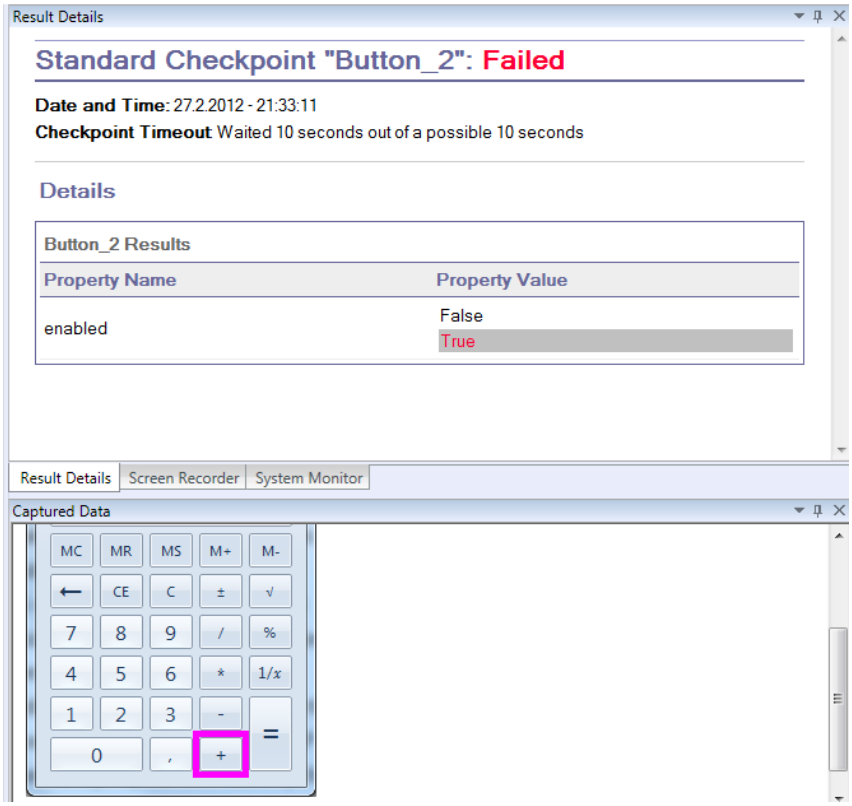
Obrázek 19 – ukázka testu v grafickém editoru

Do této krátké ukázky jsme vložili tři kontrolní body a jeden z nich očekává hodnotu, kterou test nesplní. Systém umožňuje vložit do testu kontrolní bod, který kontroluje hodnotu v objektu například v textovém poli, ale také stejně dobře umožňuje kontrolovat parametry, jako jsou výška, šířka, umístění apod.

Po spuštění testu uvidíme další velkou přednost tohoto nástroje a tím jsou výstupy výsledků. To je odvětví, kterým se předchozí Selenium IDE, téměř nezabývá. Ve výstupu je vidět v grafu jak dopadl náš test a v levém sloupci i konkrétní kontrolní body, kde došlo k chybě. A nejen to, při kliknutí přímo na chybu se dokonce zobrazí část aplikace, kde došlo k chybě a samozřejmě i očekávaná a reálná hodnota. Viz obrázky 20 a 21.



Obrázek 20 – výstup testů HP QTP



Obrázek 21 – výstup testů HP QTP

Jednu věc tomuto nástroji přeci jen vytknout můžeme. Je to používání zdrojů testovacích dat. QTP má v sobě integrovaný *TableSheet* jenže často potřebujeme data nahrávat z externích souborů. To je možné tak, že importujeme externí datový zdroj do interního. Ukázkový import je zobrazen na obrázku 22. V této oblasti je práce s externími daty pohodlnější v Selenium IDE.

```
Dim DataSheet1
Dim DataTablePath
Dim totalMatch

DataSheet2 = "Person2"
DataTablePath = "C:\temp\name.xls"

DataTable.AddSheet DataSheet2

DataTable.ImportSheet DataTablePath, DataSheet2, DataSheet2
```

Obrázek 22 – import dat v HP QTP

5.4.1 Závěr

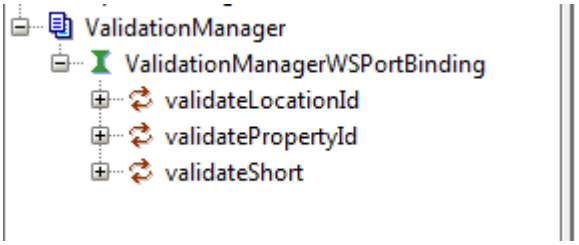
QTP je profesionální nástroj na automatické testy a kromě již zmíněné jisté složitosti při práci s externími daty, nemám co tomuto nástroji vytknout. Díky možnosti práce jak v grafickém zápisu, tak v programovém prostředí, umožní práci širší skupině uživatelů. Umožňuje také integraci s jinými HP nástroji jako je třeba HP QC což v případě používání těchto nástrojů zase o kousek usnadní práci. Kompletní dokumentace je k dispozici zde [10].

6 Nástroje pro zátěžové a vzdálené testování

6.1 SOAPui

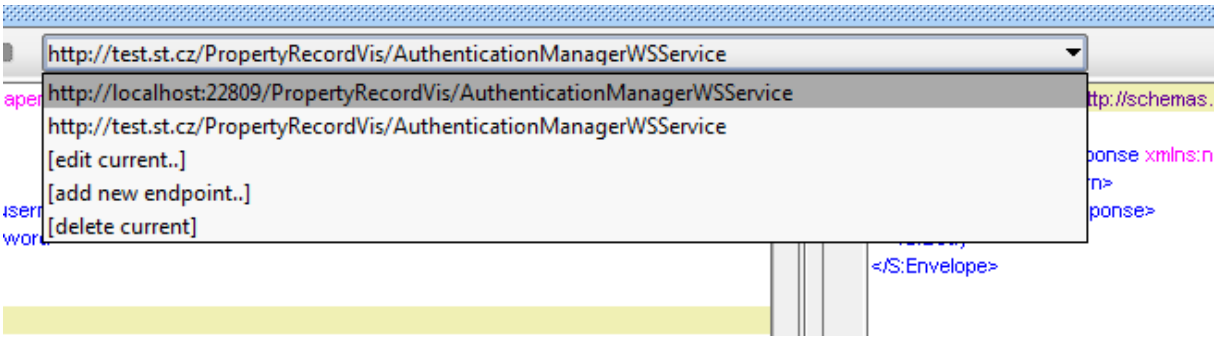
SOAPui je jeden z významných nástrojů v oblasti funkčních testů. Jak už jeho název napovídá, jeho hlavní vlastností je zasílání a příjímání SOAP zpráv. Stejně tak dobře umí pracovat s REST rozhraním. SOAPui je nabízeno ve dvou verzích, free a plná placená verze. Ve volné licenci jsme schopni vytvořit většinu základních testů. Ovšem placená verze nabízí díky grafickým nadstavbám mnohem snadnější práci. Například při přenášení parametrů mezi jednotlivými kroky. Placená verze také nabízí nástroje pro nahrávání dat z různých formátů pro automatické testy a v neposlední řadě i přehlednější výstupy výsledků. Jelikož tento nástroj je natolik obšírný zaměřím se v této práci pouze na jednotlivé zasílání SOAP zpráv, použití proměnných a vytvoření testovací sady. Nástroj dále umožňuje vytváření rozsáhlých automatických testů. Nemám tím na mysli testování uživatelského rozhraní, ale webových služeb. Dále umožňuje vytvoření zátěžových testů, kdy testujeme možnou zatížitelnost jednotlivých komponent. Vše je možné doplnit o groovy skript a napojit tak test třeba přímo na databázi.

Abychom mohli začít s testováním některé z našich komponent, je třeba znát několik věcí. Jako první je lokace WSDL v případě SOAP zpráv nebo http adresu s parametry pro REST rozhraní. Při vytváření nového projektu si SOAPui umí ze zadaného wsdl sám vygenerovat všechny operace, viz obrázek 23. V našem příkladu jsou WSDL lokace následující :
srva10rc.vsb.cz:8080/PropertyRecordVis/AuthenticationManagerWSService?wsdl
srva10rc.vsb.cz:8080/PropertyRecordVis/InsertManagerWSService?wsdl
srva10rc.vsb.cz:8080/PropertyRecordVis/ReadManagerWSService?wsdl
srva10rc.vsb.cz:8080/PropertyRecordVis/UpdateManagerWSService?wsdl
srva10rc.vsb.cz:8080/PropertyRecordVis/ValidationManagerWSService?wsdl



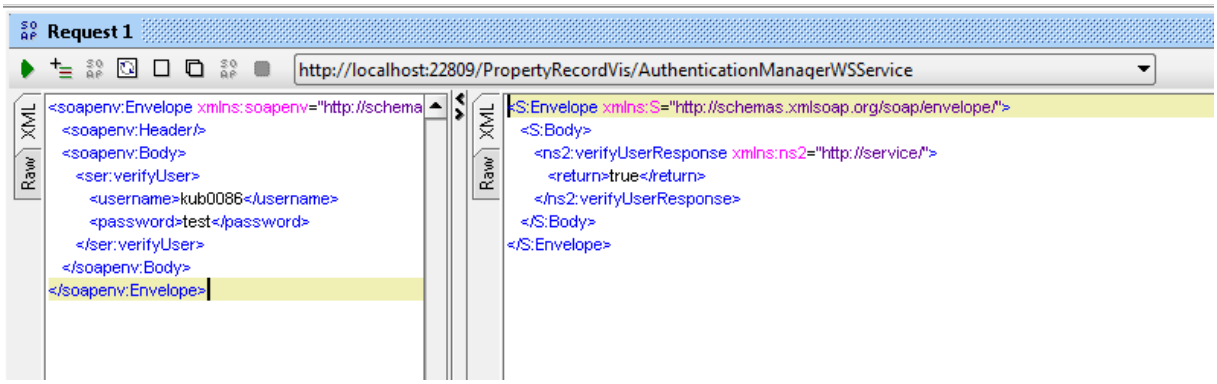
Obrázek 23 – generování metod v SOAPui

Druhou důležitou informací kterou musíme znát je endpoint naší komponenty. Systém umožňuje mít pro každý projekt nastaveno několik endpointů, takže můžeme jednoduše posílat zprávy na různá testovací prostředí bez nutnosti změny konfigurace. V našem příkladu je endpoint stejný jako wsdl pouze bez posledních znaků „?wsdl“ jak ukazuje obrázek 24.



Obrázek 24 – endpoint v SOAPui

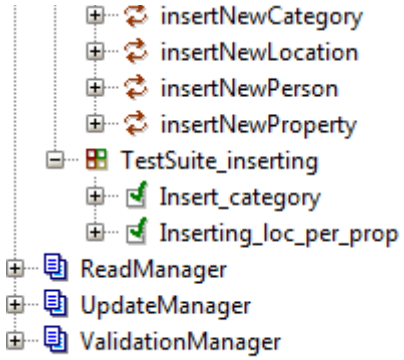
Po vyplnění xml dotazu získáme na pravé straně odpověď ve formě xml. V placené verzi je k dispozici mnohem více formátů náhledů, kdy například nemusíme vyplňovat xml dotaz, ale pouze zadáme parametry. Tyto druhy zobrazování jsou ovšem jen jakousi nadstavbou nad samotným xml dotazem. Ukázka dotazu a odpovědi je níže na obrázku 25.



Obrázek 25 – dotaz a odpověď v SOAPui

Jak jsme již zmínili SOAPui umožňuje vytváření komplexních testovacích sad, kdy každá taková testovací sada může obsahovat libovolný počet testů a každý test se může skládat z různých kroků, které si stručně popíšeme.

Do každého z našich projektů můžeme přidat testovací sadu a do něj jednotlivé testy jak je ukázáno na obrázku 26. Do jednotlivých testů následně vkládáme kroky testu.



Obrázek 26 – ukázka testovací sady v SOAPui

Test Request: tímto vložíme do testu samotnou zprávu. Tato zpráva ovšem již musí existovat v rámci našeho projektu.

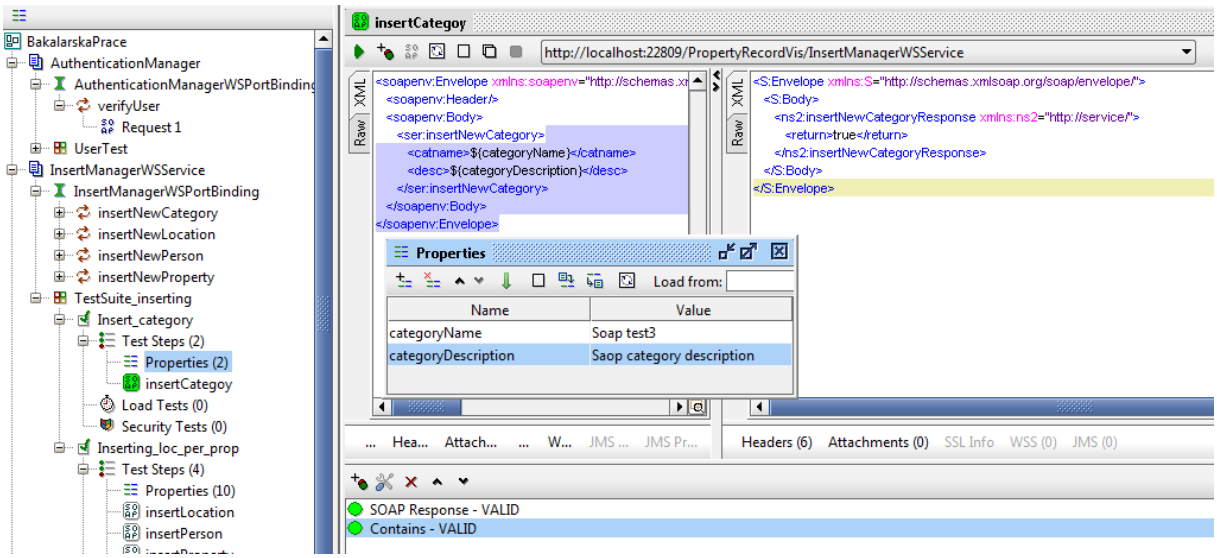
Properties: vložení vlastností a k nim přiřadit hodnoty. Jedná se o jakési proměnné, na které se ve zprávě odkazujeme za pomocí *${jméno proměnné}*.

Property transfer: tento krok slouží k převedení hodnot mezi kroky, kdy první získá nějakou hodnotu v odpovědi a druhý ji chce použít jako vstupní hodnotu v požadavku. V neplacené verzi je tento transfer poněkud složitější neboť se musí definovat ručně xml zdroj a cíl. V placené verzi je tato funkcionalita vyřešena tak, že pouze klikneme na parametr v odpovědi a dalším požadavku na parametr, kde chceme hodnotu převést.

Delay: vložení čekání pro případ, že například čekáme, než volaná komponenta provede operaci v asynchronním režimu.

Conditional Goto: tato výhybka umožňuje na základě obdržené odpovědi pokračovat určitým způsobem. Po převodu do programovacího jazyka bychom mohli říct, že je to podmínka IF.

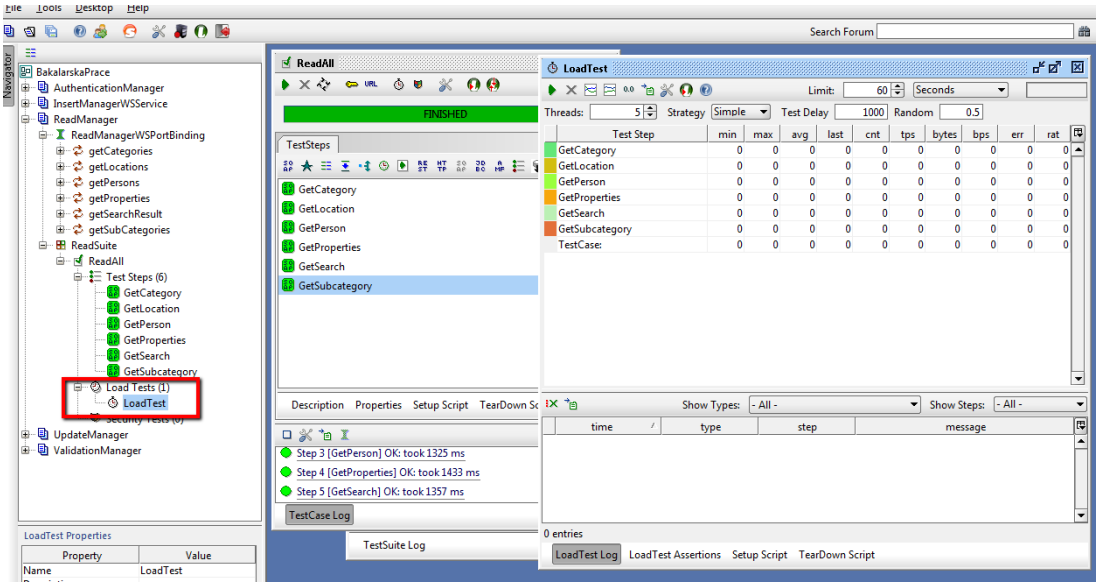
Ke každému kroku *Test Request* můžeme přidat validace výstupu *Assertions*. Díky tomu netestujeme jen, jestli test projde, ale i jestli nám komponenta vrátí očekávány výstup. Spouštět následně můžeme jednotlivé testy každý zvlášť nebo celou testovací sadu, která provede spuštění všech testů. Na obrázku číslo 27, vidíte úspěšně provedený test.



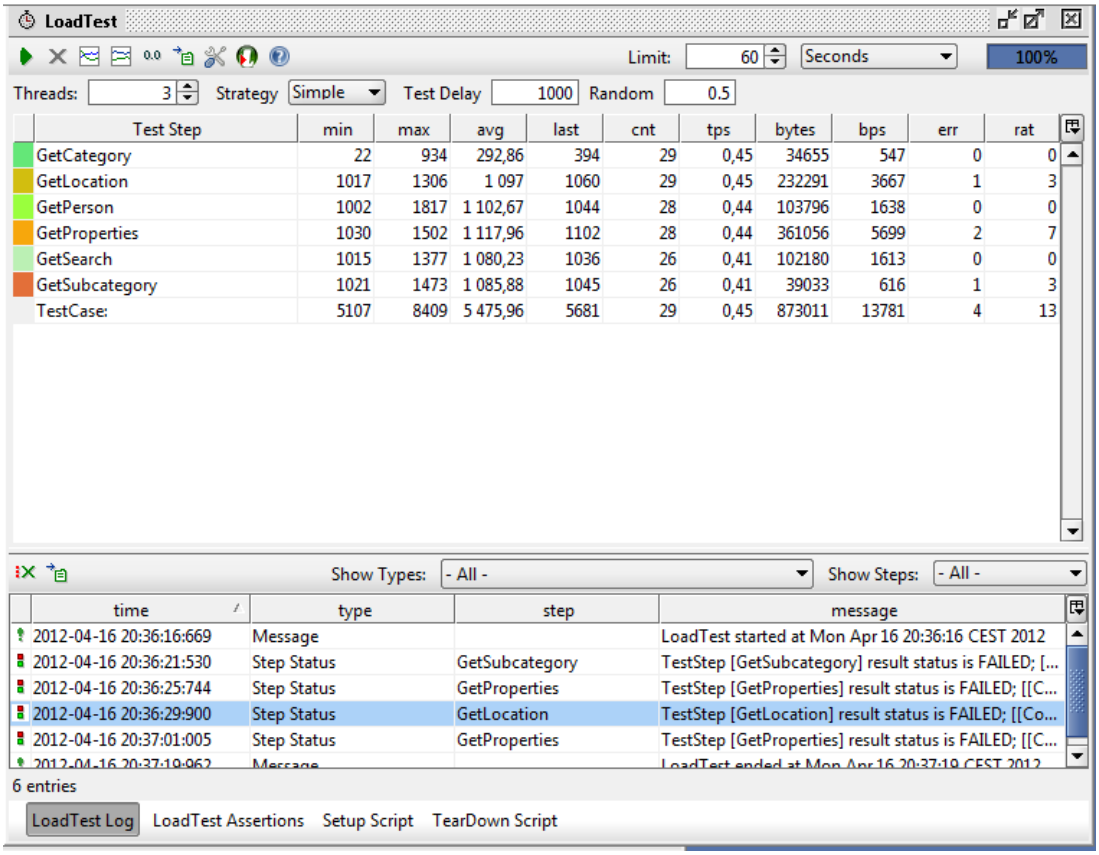
Obrázek 27 – ukázka testu v SOAPui

6.1.1 Zátěžový test

Ke každé vytvořené testovací sadě, máme možnost přidat zátěžový test označený jako Load test. Vytvořením takového to testu je jednoduché, protože při přidání nám SOAPui převede celý funkční test do zátěžového, aniž bychom museli cokoliv vytvářet ručně. Jediné co musíme nastavit je počet vláken, doba trvaní apod. Ukázka takového to testu je na obrázku 28. Je zde ukázáno vytvoření zátěžového testu z šesti kroků. V případě, že budeme mít v testu i transfer proměnných, použití groovy skriptů apod. tak se nám tyto koky nebudou započítávat do konečných statistik. Výsledky testů můžeme vidět okamžitě pomocí grafu a případné chyby se vypisují na spodní straně testu jak je vidět na obrázku číslo 28. Celý výsledek je možné i exportovat do xml souboru. Výsledky, ale pouze kopírují tabulku z obrázku 27.



Obrázek 27 – ukázka zátěžového testu v SOAPui



Obrázek 28 – výsledky zátěžových testů v SOAPui

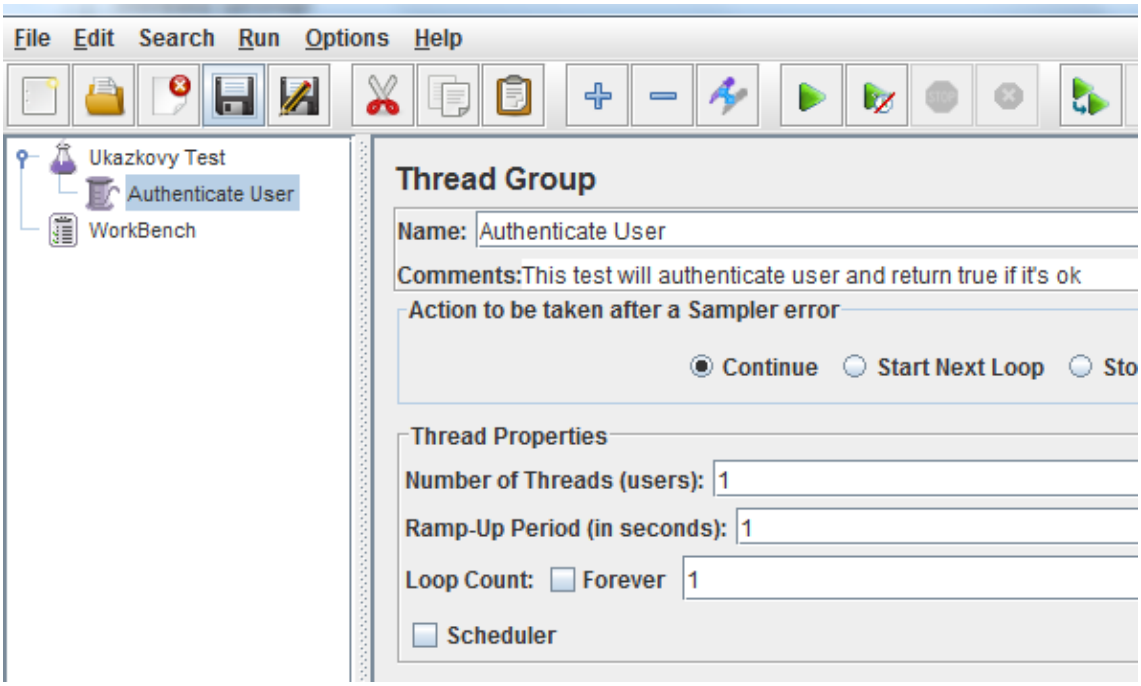
6.2 Apache JMeter

JMeter je jeden z nástrojů zaměřený na výkonnostní testování. Využívá se k simulaci víceuživatelského přístupu a tím testuje zatížitelnost testovaného objektu. JMeter může testovat několik typů servrů:

- Web – http, HTTPS
- SOAP
- Databáze – JDBC
- LDAP
- JMS
- Email – POP3(s) a IMAP(s)

U testů nám dovoluje nastavit celou škálu parametrů jako například počet vláken (počet virtuálních uživatelů), náběh těchto vláken (jestli se má celá zátěž spustit najednou nebo se má spouštět postupně s daným časovým intervalem). V ukázkovém testu si projdeme základní nastavení a vytvoření testu.

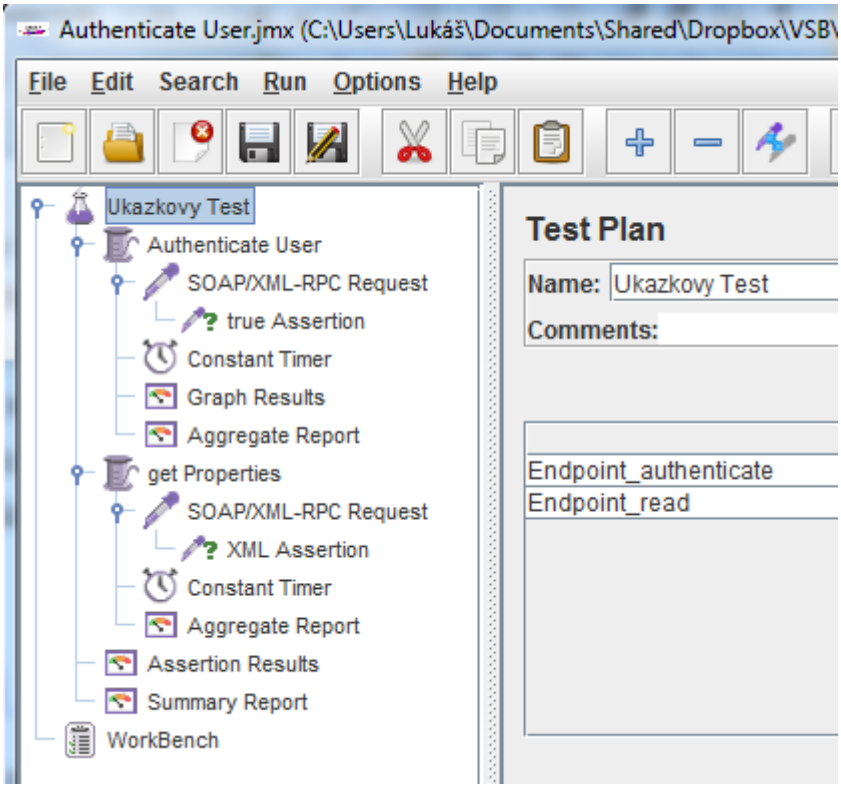
Přímo v plánu testů je možnost si nastavit globální proměnné a je to vhodné místo pro nastavení endpointů a adres serverů. Na obrázku 29 je ukázaná skupina vláken, kde nastavujeme samotnou zátěž. Jednotlivé nastavení není třeba popisovat, protože grafické rozhraní programu je opravdu srozumitelné.



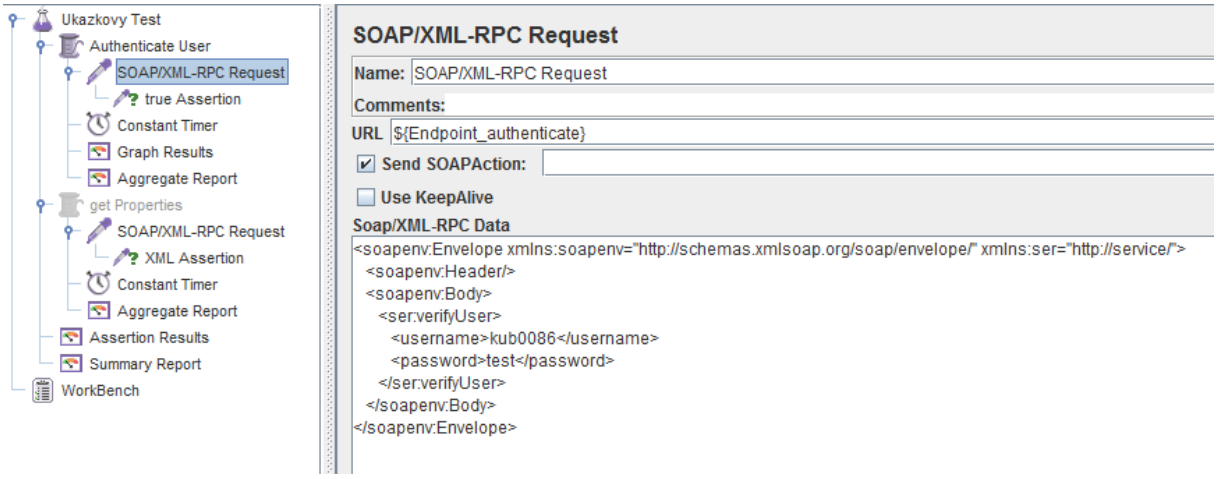
Obrázek 29 – nastavení vláken v aplikaci JMeter

Poté je třeba vytvořit dotaz, který budeme na příslušný server zasílat. V mém případě je to SOAP, který je v testu reprezentovaný jako Sampler – SOAP/XML-RPC request. Samozřejmě, že v rámci

jednoho test plánu je možno mít větší počet testů, kdy každý test může obsahovat libovolný počet požadavků. Je možno kombinovat například SOAP s jinými dle potřeb testů. Na každý požadavek je možné navázat *Assertion* ověřování správnosti výstupu. Těch může být i více, v závislosti na potřebě. Ukázkový test plán může vypadat třeba jako na obrázku 30 a 31.



Obrázek 30 – ukázka testu v JMeter



Obrázek 31 – ukázka zprávy v JMeter

Kdy potom stačí vložit validaci odpovědi, například jako já XPATH Assertion s parametrem */Envelope/Body/verifyUserResponse/return="true"*

Ke každému testu máme možnost přiřadit dle potřeb libovolný počet funkcionalit, jako jsou grafy zobrazující průběh testů, nebo různé jiné druhy reportů, protože právě zobrazení správných výsledků je u výkonnostních testů nadmíru důležité. Možnost přidat k testu nebo k celému plánu (monitoruje výsledky všech testů dohromady) různé druhy monitorování je velké plus, avšak na druhou stranu nástroj již neřeší co s výstupem dále dělat. Máme možnost exportovat výsledky do souborů, ale bohužel jen v textové podobě. Často je zapotřebí vypočítat kolik a jakých požadavků neprošlo, nebo se vrátilo s chybou. Takže v případě, že potřebujeme výsledky lépe formulovat nebo zobrazovat, tak JMeter nám rozhodně nepomůže. Bude pak jen na nás obsáhlé logy zpracovat a vyvodit z nich závěry. Jako klad bychom mohli vyzdvihnout velkou variabilitu nastavení testů a spouštění zátěže. Ať už jde o počet, vláken, počet iterací nebo jestli se mají testy spouštět najednou nebo postupně. Kladem je určitě i jednoduchý způsob jak nahrát testovací data z externího souboru. Slouží pro to CSV Data Set Config.

Při těchto testech je třeba si uvědomit jednu zásadní věc. A tím jsou omezení vlastního počítače. Udělali jsme proto malý test. Vytvořili jsme SOAP test, který ověřuje uživatele v systému. Nejedná se o velký přenos dat a vzhledem k jednoduchosti implementace by případné prodlevy neměla způsobit testovaná komponenta. Tato komponenta byla umístěna na virtuálním školním serveru a test byl spouštěn z lokálního počítače přes VPN připojení. Pro zajištění propustnosti linek byl test spuštěn v nočních hodinách. Ve všech případech, je počet iterací 50. Výsledky ukazuje obrázek 31.

200	**Průměr**	470ms
	Chybovost	0,00%
300	**Průměr**	882ms
	Chybovost	0,00%
400	**Průměr**	1298ms
	Chybovost	0,11%
500	**Průměr**	1707ms
	Chybovost	0,27%

Obrázek 32 – výsledky pokusu stability

V případě 200 vláken byl průměrný čas vyřízení jednoho požadavku 470ms a 0% chybovosti. Během testu bylo zatížení lokálního počítače někde okolo 60%. V druhém případě již bylo zatížení počítače okolo 90% a jde vidět, že i průměrný čas na vyřízení požadavku stoupl. V posledních dvou případech bylo zatížení počítače na maximu a je vidět, že se to projevilo jak na čase vyřízení požadavku, tak na chybovosti. Z toho je patrné, že špatně určená zátěž nám může zkreslit výsledky testů a to z důvodu výkonnostního limitu počítače, ze kterého je test spouštěn. V případě, že bychom přenášeli větší množství dat, je třeba započítat i zatížení sítě. Takže jestli máme možnost, je lepší pustit testy z více počítačů při nižší zátěži. Což je dnes i běžná praxe pro tento druh testování.

7 Nástroje pro podporu vývoje a testování

7.1 Motivace

Vývoj softwaru není triviální záležitost a zahrnuje dnes celou řadu jednotlivých kroků a úkolů. Již po přečtení kapitoly o SCM nám musí být jasné, že bez kvalitních podpůrných nástrojů se neobejdeme. Potřebujeme nástroje, které nám umožní rozdělit jednotlivé části vyvíjeného systému na co nejmenší dílčí úkony. Zároveň potřebujeme monitorovat průběh vývoje ať už celkově nebo v jednotlivých iteracích. To nám dává možnosti efektivně rozdělovat práci, monitorovat vývoj a odhalovat úzká místa. To vše a mnohem více je základem pro úspěšné řízení vývoje softwaru. K tomu všemu dnes již existuje několik nástrojů a my si tady dva z nich popíšeme.

7.2 Jira

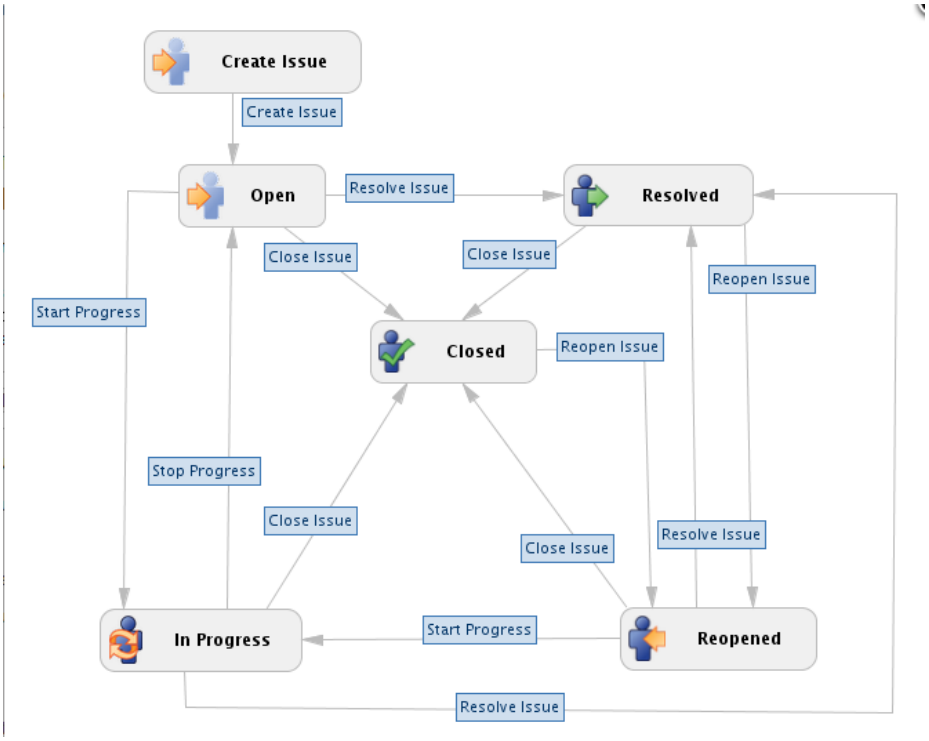
Nástroj JIRA od společnosti Attlassian [11] patří k nejpoužívanějším nástrojům z oblasti řízení projektů při vývoji softwaru. Tento nástroj je placený a cena se odvíjí od množství uživatelů. Pro orientaci uvedeme, že do 10 uživatelů je cena pouze $10/měsíc a při 2000 uživatelů to je $1000/měsíc. Již z toho je jasné, že se nejedná o příliš drahou aplikaci vzhledem k ostatním projektovým nákladům. JIRA poskytuje obrovskou flexibilitu a záleží na daném projektu, jak bude nakonfigurován. Každý projekt může mít jiný průběh, budou na něm pracovat jiní lidé, je třeba, aby procházel jinými statusy apod. JIRA to vše umožňuje a pro nově vytvářený projekt je možné vše nadefinovat.

Při vytváření nového projektu můžeme definovat tyto základní vlastnosti.

1) Typ záznamu
   a) Bug: jedná se o nalezenou chybu při testování aplikace nebo během jiného životního cyklu vyvíjení
   b) Epic: jde o velkou část nové implementace systému. Epic je následně dělen na případné pod části, které jsou tvořeny jednotlivými User story
   c) Improvement: vylepšení již existující funkcionality aplikace
   d) New Feature: nová funkce vyvíjeného systému
   e) Task: jde pouze o nespecifikovanou práci například analýzu, vytvoření dokumentace a jiné.
   f) Story: jde o user story, kdy je best practice mít pro každý testovací případ jednu user story.

   Pro každý takovýto požadavek můžeme specifikovat i jednotlivá políčka, která chceme, aby byla zobrazena a vyplněna. Jde například o popis, verze sestavení, ve kterém byl problém nalezen, komponenta a jiné.

2) Workflow: pro projekt můžeme vytvořit specifický průběh požadavků. Možná ukázka je na obrázku číslo 33.

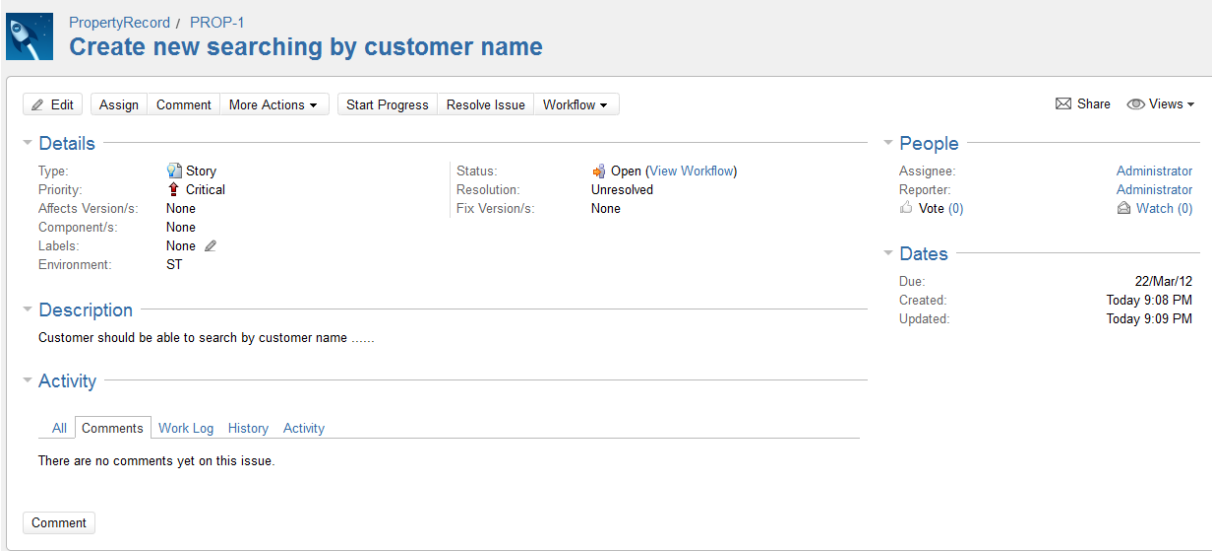


Obrázek 33 – ukázka možného průběhu stavů požadavků v nástroji Jira

3) People: zde je možné nastavit tým lidí, kteří budou na projektu pracovat a přiřadit jim role v rámci projektu. Samozřejmě, že jde určitým lidem práci s projektem zakázat nebo omezit.

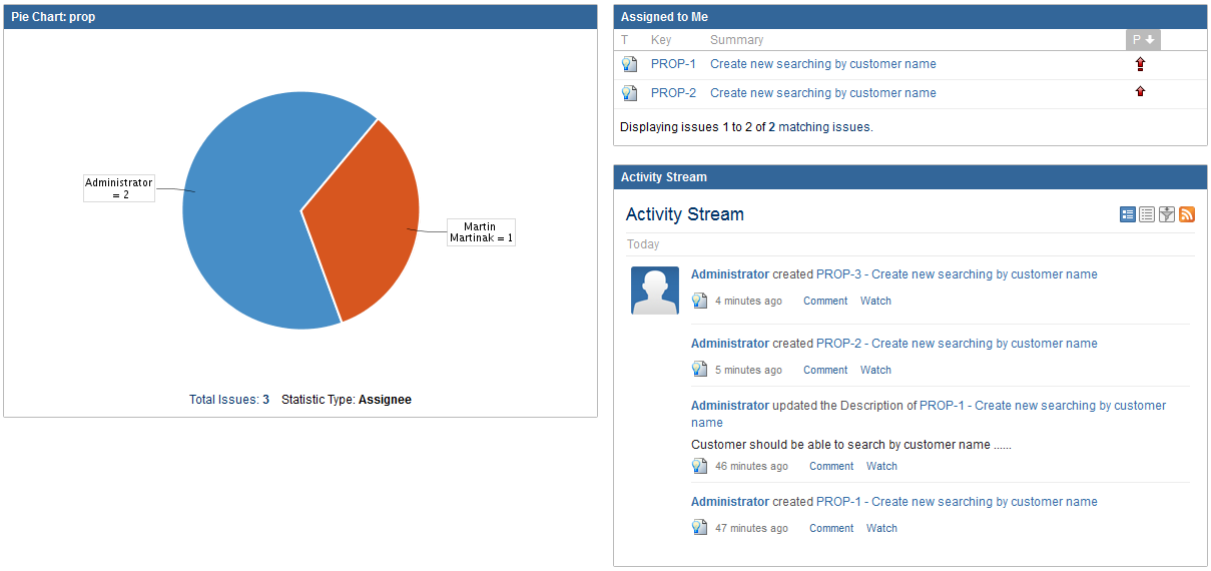
Toto bylo jen pár základních věcí, protože možností máme mnohem více. Ať už jde o konfigurace zasílání automatických emailů, synchronizaci s GIT, definovaní vlastních polí apod.

Vytvořený požadavek je zobrazen na obrázku číslo 34.



Obrázek 343 – ukázkový požadavek v nástroji Jira

Samotné vytváření požadavků a reportování chyb by samo o sobě nebylo k ničemu, kdyby systém neumožňoval vše sledovat a přehledně zobrazovat. V Jira má každý uživatel svůj Dashboard, což by se dalo přeložit, jako jakási pracovní plocha, na kterou si vkládá různé monitorovací nástroje. Může se jednat o předpřipravené výstupy například, jak je zobrazeno na obrázku 35.



Obrázek 35 – ukázková pracovní plocha v nástroji Jira

Jira nabízí veliké množství různých zobrazovacích a sledovacích nástrojů, které si na svou pracovní plochu můžeme vložit a sledovat tak naši práci, průběh práce na projektu, sledovat kolik ještě zbývá udělat požadavků apod. Vše je svázáno s filtry, které je možno vytvořit v záložce Issues. Nejde o nic jiného než o jakýsi pseudo SQL dotaz, který vyhledá zadaný požadavek. V případě, že odpovídá požadovanému výsledku, můžeme si jej uložit jako filtr a ten pak svázat s monitorovací aplikací na své pracovní ploše. Dotaz filtru by mohl vypadat třeba takto: *project = PropertyRecord AND assignee*

= *tac9899 AND status != Closed*. Tento dotaz vyhledá všechny nezavřené požadavky, které jsou připsané uživateli *tac9899* a spadají do projektu *PropertyRecord*. Velkou výhodou pracovních ploch je veliká variabilita a každý uživatel si ji může přizpůsobit tak, aby se mu s ní dobře pracovalo a měl všechny potřebné informace tam, kde je mít chce. Dalším velkým plusem je fakt, že pracovních ploch můžeme mít několik a na každé zobrazovat jiné informace nebo informace k různým projektům. Jakoukoliv pracovní plochu můžeme sdílet a to buď se všemi skupinami, nebo jednotlivými uživateli. Díky tomuto můžeme vytvořit sdílené pracovní plochy v rámci projektu a zajistit tak, že všichni, kteří pracují na jednom projektu, uvidí kromě svých informací také sdílené informace.

Jira tak eviduje a archivuje celý životní cyklus vyvíjené aplikace. Jednotlivé části vývoje dovoluje rozbít a rozdělit na malé atomické kroky a ty potom připisovat jednotlivým členům týmu k práci. Přitom všem umí udržovat přehled o již hotové práci a práci, která musí být ještě dohotovena. V případě nalezení chyb ve fázi testování umožňuje vytvoření specifického požadavku a jeho svázání s testovanou části. Přehledně tak zpracovává chybovost jednotlivých částí, z kterých je aplikace tvořena. Díky tomu umožňuje snazší rozhodování například o tom, jestli je kvalita dostačující pro uvolnění do produkce.

7.3 HP QC

HP Quality Center je stejně tak jako JIRA nástroj pro podporu vývoje a testování programů a aplikací. A stejně tak i v QC máme projekty, požadavky Requirements, v případě nalezení chyby vytvoření odpovídajícího požadavku Bug, filtrování jednotlivých požadavků na základě pracovních skupin apod.

Oproti předchozímu nástroji má QC něco navíc a to je řekněme databáze testů pro manuální testování. Můžeme si zde vytvořit kompletní testovací projekty, které mohou obsahovat testovací sady. Pro každou takovou sadu následně vytvořit testovací případy a u nich specifikovat jednotlivé kroky. Tato struktura nám dává možnost efektivně využívat databázi testů pro libovolný počet zákazníků, testovacích sad apod. Následně při testování tyto testy spustit, procházet předem vytvořenými kroky a pouze zaškrtávat jestli test prošel nebo ne. Samozřejmě při nalezené chybě můžeme ihned vytvořit příslušný požadavek na její odstranění. Na konci tak máme velice šikovný přehled o tom, které testy prošly a u kterých byly nalezeny nedostatky. Tato funkcionalita je k nezaplacení při regresním testování, kdy je třeba mít někde uložené sady regresních testů. Navíc můžeme mít pro libovolnou část systému uloženo více regresních testů a spouštět ty, které zrovna potřebujeme. Takže například mějme zákazníka, pro kterého vytvoříme v testovacím plánu testovací projekt *A*. Pod tímto projektem si vytvoříme několik testovacích sad, například *test rozhraní A* a *test rozhraní B*. V rámci každé této testovací sady pak vytvoříme několik sad pro regresní testování, například regresní test pro krizové situace, regresní test pro nedostatek času, který bude obsahovat jen základní kritické části apod. Následně si spustíme test, který zrovna potřebujeme. Po výběru potřebné testovací sady se nám rozbalí seznam jednotlivých testovacích případů pro námi vybranou testovací sadu například regresní test. Kliknutím na testovací případ spustíme test. Spuštěním testu se otevře nové okno, ve kterém bude seznam všech kroků pro daný testovací případ. Manuálně provedeme test a zaškrtneme, jestli byl výsledek daného kroku úspěšný či nikoliv. Následně přejdeme na další krok až do otestování celého

testovacího případu. V praxi může dané testy vytvořit test manažer a vytvořením požadavku, což je podobné jako v Jira nástroji, připsat tento požadavek určitému člověku. Výhoda těchto databází testů je ta, že testy jsou vytvořeny obecně a můžou, ale nemusí být svázány s uvolněním do produkce.

Například test přihlášení uživatele, by mohl mít tyto Kroky. 1. Zadání uživatelského jména a hesla. 2. Kliknutí na přihlášení, kde u druhého kroku bychom do očekávaného výsledku zapsali, že uživatel byl úspěšně přihlášený.

7.4 Porovnání Jira a HP QC

Jestli bychom měli tyto dva nástroje porovnat, tak JIRA je více zaměřená na životní cyklus vývoje aplikací, kde neřeší pracovní prostor pro samotné testování. Kdežto QC je kromě životního cyklu vývoje zaměřen i na podporu testování. Jde o jednotné místo, kde můžeme mít uloženy všechny potřebné testy, dokumentaci k testům a výsledky testů v jednotlivých iteracích vývoje. Samozřejmě, že stejně jako v nástroji QTP od tohoto výrobce, i zde je velice dobře zpracováno grafické zobrazování výsledků. QC obsahuje i celou řadu management nástrojů jako jsou například přehledy již provedené práce na požadavku, iteraci nebo projektu. Umožňuje také spravovat lidské zdroje a tím napomáhá k přesnějšímu stanovení času do ukončení vývoje a testování. Umožňuje taktéž exportovat jednotlivé části do textové podoby a tak mít k dispozici i kompletní dokumentaci k projektu. K tomuto nástroji je velice podrobně zpracována dokumentace [12]. Tento nástroj má ovšem nižší uživatelskou flexibilitu než Jira, protože zde není možné si vše přizpůsobit dle našich konkrétních potřeb a požadavků. A druhou devizou je jeho cena, která je o dost vyšší. Na druhou stranu pro oblast testování nabízí velkou podporu, která je navíc sdílená na jednom místě a v týmu se tedy odstraňuje redundantní práce. Z tohoto pohledu se může zdát, že HP QC má oproti Jira hodně navrch. Ale je třeba si uvědomit, že oba tyto nástroje obsahují vše pro podporu vývoje softwaru a často právě možnost flexibilní úpravy pro různé projekty je velkým přínosem, který je ještě umocněn nižší cenou.

8 Celkové zhodnocení

Při výběru vhodného nástroje je třeba zhodnotit potřeby projektu a naše vlastní možnosti. Našimi vlastními možnostmi je myšleno to, že si nevybereme pro automatické testy WebDriver v případě, že nikdo v týmu neumí programovat. Pro většinu placených nástrojů existuje i Open source alternativa, ale je to za cenu kompromisů. Jeden znatelný příklad je právě v automatickém testování a reportů u HP QTP a Selenium IDE. Placené nástroje obsahují navíc často funkcionality, které nahradíme jen těžko. Například u zmíněného QTP je to testování desktopových aplikací. Nebo u HP QC je to úložiště a sdílení testů. Na trhu samozřejmě existuje celá řada nástrojů a QTP není jediný, který umí testovat desktopové aplikace. Je tady například taktéž placený nástroj TestComplete. Stejně tak jako jsme si ukázali SOAP zátěžový test v JMeter můžeme vytvořit tento test i v SoapUi. Ale nejde jen o to, co v čem můžeme či nemůžeme vytvořit. Jde taky o to, jak přehledný test bude a jak nám dokáže zpracovat výstup. SoapUi sice dovede pouštět výkonnostní testy, ale má trochu problém s využíváním paměti počítače při těchto testech. Takže se jednoduše může stát, že dojde k přetečení a náš test nebude úspěšný. Tyto problémy sice lze řešit a na internetu je spousta návodů, ale nebylo by třeba lepší použít nástroj přímo vytvořený pro tyto typy testů?

8.1 Porovnání jednotlivých nástrojů

Pokusíme se zde udělat základní porovnání nástrojů, které je částečně ovlivněno subjektivní zkušeností při práci s nimi.

8.1.1 Tvorba testovacích dat

Zde je čistě na nás jaký jazyk či platformu pro tvorbu testovacích dat použijeme. Je třeba zvážit požadavky daného projektu, platformu, ve které se projekt vyvíjí. Ve výsledku je jedno, zda jsou skripty psané v Java nebo Ruby, protože vše splní svůj účel. Už ale není jedno, kolik času nám zabere tvorba a následná údržba skriptů, protože je třeba si uvědomit, že to jsou pomocné skripty a měly by nám práci usnadnit a ne komplikovat. Je třeba zvolit takový jazyk, který se nám nejsnadněji integruje v prostředí projektu a zároveň je v týmu několik lidí, kteří jej aspoň základně ovládají. Výhoda vlastních skriptů spočívá v možnosti přizpůsobení přesně pro naše požadavky. Druhou výhodou bude cena. Protože například u zmíněného DTM nástroje je u Entrprice edice cena za jednu licenci $349. Na druhou stranu je i velká řada open source nástrojů, které můžeme najít například v odkazu [13].

8.1.2 Automatické testování

V této části jsme si ukázali tři nástroje, Selenium IDE, WebDriver a HP QTP. V případě, že bychom na projektu nemuseli řešit finance a mohli si dovolit jakýkoliv nástroj, tak QTP je určitě správná volba. Testy jdou kvalitně nahrát a jednoduše upravit v grafickém zápisu a tím pádem není nutná znalost programovacího jazyka VBS. Umí testovat jak webové tak desktopové aplikace. Navíc má velice pěkně řešeny výstupy testů, jak jsme si již ukázali. Bohužel se finance řeší často a tak porovnáme i dva nástroje pro webové aplikace. Selenium IDE je vhodné v případě, že nechceme mít testy závislé na znalosti programovacích jazyků. Tvorba testů je jednoduchá a s doplňky tvoří docela slušný nástroj. Při spouštění testů ovšem narazíme na různá úskalí, která mohou práci značně ztížit. Například příkaz *waitFor xxxxx* nezafunguje vždy na sto procent a tak často je vhodnější použít příkaz *pause* s dostatečným intervalem. Zvýšíme tím tak stabilitu testu, ale to vše na úkor doby, po kterou test běží. Při testování složitějších aplikací se stává, že selže komunikace mezi nástrojem a webovým prohlížečem a celý test nám pak spadne. Tento problém nastává u složitějších aplikací při použití CMS (Content Management System). Na druhou stranu WebDriver vyžaduje znalost jednoho z programovacích jazyků (Java, C#, Ruby, Python, php, Perl), kdy ne vždy je u všech jazyků stejná podpora. Takže například některé funkcionality, které nefungují pod Ruby, budou fungovat za použití Java. Stabilita testů, zde závisí na dovednosti toho, kdo test píše. Záleží na něm, jak bude řešit výjimky, jak chce zpracovávat výstupy, jestli napíše test jako Unit test, nebo jako nezávislý skript. Je zde možnost si vytvořit vlastní knihovny nebo Object Library. Takže v případě, že máme znalosti s programovacími jazyky, tak WebDriver bude asi tou správnou cestou. Ale jak jsme se již zmínili, záleží hodně na tom, co testujeme, protože jestli je naše testovaná aplikace jednoduchá, nebo chceme zautomatizovat jen část, tak Selenium IDE nemusí být špatnou volbou.

8.1.3 Testování vzdálených komponent

V případě, že testujeme i vzdálené komponenty tak se bez SOAPui nástroje neobejdete. Pro základní práci a tím myslíme posílání SOAP nebo REST dotazů, vytváření automatických testů nebo celých test plánů nám stačí i neplacená verze. Je to ovšem za cenu složitější práce například v již zmíněné části *property transfer*. V tomto případě placená verze nabízí zjednodušení práce, *security testing*, větší výběr zobrazení odpovědí na dotazy a vylepšené reportování výsledků.

8.1.4 Apache JMmeter

V této práci jej můžeme porovnat jen se zátěžovým testem v SOAPui. V případě, že nepotřebujeme jiné než SOAP nebo REST testy postačí nám SOAPui. Dokonce je v tomto případě i vhodnější, protože tím zároveň budeme mít i sadu jednotlivých testů, které nemusíme spouštět v zátěžovém testu, ale libovolně jak potřebujeme. V případě, kdy ovšem potřebujeme jiný požadavek než SOAP nebo

REST tak je JMeter velkým pomocníkem. Díky své velké variabilitě možných testů pokrývá snad všechny možné případy, jaké mohou nastat. Nesmíme však zapomínat na již zmíněné omezení počítače a sítě, z kterého se testy spouštějí.

8.1.5 Přehled vlastností a subjektivní zhodnocení

	Selenium IDE	WebDriver	HP QTP	SOAPui	JMeter
Automatické testování	X	X	X	X	
Výkonnostní testování				X	X
Testování komponent				X	X
Webové aplikace	X	X	X		X
Desktop aplikace			X		
Licence	F	F	P	F/P	F
Nutná znalost programování	V	X	V		
Vícejazykovost		X			

Subjektivní hodnocení	50%	90%	70%	80%	70%

Tabulka 1- porovnání nástrojů

V – znalost není podmínkou, ale značnou výhodou
F – volná licence
P – placená licence

Osobní hodnocení nástrojů jsme provedli na základě jednoduchosti použití. Jinými slovy jak rychle jsme daný nástroj pochopili, dovedli použít a také jak má řešené jednotlivé funkcionality.

Selenium IDE, dostal 50% a to z důvodu nutnosti java skript knihoven, bez kterých má omezené možnosti. Některé prvky pseudo kódu nefungují tak jak by uživatel očekával a chybí vhodný export výsledků. Jinak integrace tohoto nástroje do webového prohlížeče a množství více či méně užitečných doplňků je velké plus.

WebDriver má téměř plné hodnocení, protože je možná velká variabilita díky tomu, že je psán přímo v programovacím jazyku. Pro člověka znalého alespoň základů programování je naučení se práce s WebDriver otázkou několika málo hodin. Co můžeme vytknout je neúplná podpora ve všech jazycích, i když i to je pochopitelné, kdy největší podpora je věnována právě jazyku Java.
HP QTP tento nástroj jsme měli na zkoušku jako trial verzi po dobu jednoho měsíce. Nástroj má velkou řadu funkcionalit a tak pro plné využití je třeba věnovat dost času studiu jednotlivých funkcí. I přes to, ovšem vytvoření prvních jednoduchých testů je otázkou několika málo minut a to především

díky dobře zpracovanému grafickému zobrazení. Pro pokročilejší funkce je třeba věnovat čas dokumentaci.

V případě, že chceme pouze posílat a přijímat SOAP či REST, tak není třeba žádného hlubšího studia tohoto nástroje. Tyto základní funkce jsou ovládány velice intuitivním způsobem. V případě, že chceme vytvořit celé testy či sady testů tak je placená verze k nezaplacení. A pro složitější testy se bez této verze neobejdete.

JMeter pro zátěžové testování je nástroj s kterým není těžké se naučit pracovat, i když je třeba se mu věnovat a projít si některé tutoriály či dokumentace. Těch je ale na internetu dostatek a vytvořit první test nebyl velký časově náročný problém.

8.2 Shrnutí

V případě že chceme vytvářet automatické testy pro webové aplikace tak WebDriver je vhodná volba. Testy si můžeme vytvořit naprosto libovolně, dle potřeb, integrovat je s vývojovým prostředím, spouštět samostatně či jako unit testy. Jestli testujeme jinou než webovou aplikaci tak nezbývá nic jiného než HP QTP nebo se poohlédnout po jiném nástroji. Pro zátěžové testy je možné použít SOAPui, ale jen v případě SOAP či REST testů. Pro složitější zátěžové testy je JMeter správnou volbou, protože je pro tyto typy testů vytvořen.
Samozřejmě, že tato práce nehodnotí mnoho dalších jiných nástrojů, kterých je dneska celá řada. Například předchůdce WebDriveru je Selenium RC, kdy jsme před samotným testem museli spustit Remote server a ten zpracovával naše požadavky. Tato platforma je již bez podpory a vzhledem k mnoha verzím Remote serveru je vývoj značně komplikovaný a rozhodně jej není možné doporučit. U výkonnostních testů to může být například LoadUI či loadrunner od společnosti HP. Nechceme zde a ani nemůžeme říci, jestli ten či onen nástroj je nevhodné používat. Všechny mají své silné a slabé stránky a je vždy třeba zvážit požadavky a možnosti projektu, pro který budou použity.

Literatura

[1] Testování softwaru. *Wikipedia.org* [online]. 20.2.2012 [cit. 2012-04-08]. Dostupné z: http://cs.wikipedia.org/wiki/Testov%C3%A1n%C3%AD_softwaru

[2] KANER, Cem, Jack FALK a Hung Quoc NGUYEN. *Testing computer software*. 2nd ed. New York: Wiley Computer Publishing, c1999, 480 s. ISBN 04-713-5846-0.

[3] Software Testing Guide Book. In: *Software Testing Guide Book* [online]. 6.4.2004, 3.7.2004 [cit. 2012-04-20]. Dostupné z: http://www.scribd.com/doc/8769556/Software-Testing-Guide-Book-Part-1

[4] J. MILLIGAN, Tom a David E. BELLAGIO. *What Is Software Configuration Management?* [online]. 2005 [cit. 2012-04-08]. Dostupné z: http://www.informit.com/articles/article.aspx?p=390813

[5] BELLAGIO, David E, Tom J MILLIGAN a Brian A WHITE. *Software configuration management strategies and IBM Rational ClearCase*: *a practical introduction*. 2nd ed. Upper Saddle River, NJ: IBM Press, c2005, 346 s. ISBN 03-212-0019-5.

[6] *Generatedata* [online]. n/a [cit. 2012-04-16]. Dostupné z: www.generatedata.com

[7] *Visual studio Data Generátor*. n/a. Dostupné z: http://www.dotnetfunda.com/articles/article1159-visual-studio-data-generation-plan.aspx

[8] Sqledit. *Sqledit* [online]. [cit. 2012-04-16]. Dostupné z: http://www.sqledit.com/index.html

[9] SeleniumHQ. SELENIUM.ORG. *Selenium HQ* [online]. n/a [cit. 2012-04-08]. Dostupné z: http://seleniumhq.org/projects/ide/

[10] QTP user guide. In: *Wordpress* [online]. 31.5.2004, 1.11.2004 [cit. 2012-04-08]. Dostupné z: http://thinh1808.files.wordpress.com/2008/03/qtp-tutorial.pdf

[11] *Jira* [online]. n/a [cit. 2012-04-20]. Dostupné z: http://www.atlassian.com/software/jira/overview

[12] QC Tutorial. In: *QC-10-Tutorial* [online]. n/a [cit. 2012-04-20]. Dostupné z: http://www.wiziq.com/tutorial/163404-QC-10-Tutorial-pdf

[13] Webresourcesdepot. *Data Generators* [online]. n/a [cit. 2012-04-20]. Dostupné z: http://www.webresourcesdepot.com/test-sample-data-generators/

Přílohy

Příloha na CD/DVD

- Selenium IDE – JS knihovny pro Selenium IDE, csv soubory pro testovací data, samotný test suite
- TestDataJava – Java projekt pro tvorbu testovacích dat
- WebDriver – csv soubor pro testovací data, podpůrná knihovna „WebDriverToData“ a samotný test „TestSuitePropertyRecord“
- SOAPui – uložené projekty pro SOAPui nástroj
- PropertyRecordVis – testovaná aplikace, která obsahuje web a implementované WS pro SOAPui
- JavaDB – databáze použitá v testované aplikaci